4

ERL-0505-GD

AR-005-990

AUSTRALIA

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
SALISBURY

ELECTRONICS RESEARCH LABORATORY

SOUTH AUSTRALIA

GENERAL DOCUMENT

ERL-0505-GD

GRAPH MATCH PROGRAM

MARK T. BARRINGTON

MARCH 1990

CONDITIONS OF RELEASE AND DISPOSAL

UNCLASSIFIED

ERL-0505-GD

AR-005-990

DSTO AUSTRALIA

ELECTRONICS RESEARCH LABORATORY

GENERAL DOCUMENT
ERL-0505-GD

GRAPH MATCH PROGRAM

Mark T. Barrington

ABSTRACT (U)

Described herein is a program for manipulating and comparing graphs of radar cross-sections. The size and position of one graph relative to another can be changed and the two graphs combined to produce a new graph which represents an average of the two. The program can process data sets of varying numbers of data values. Graph Match was written using Apple Macintosh SE and II computers and Turbo Pascal as the programming language.

Postal Address: Director, Electronics Research Laboratory,
PO Box 1600, Salisbury, South Australia, 5108.

CONTENTS

1 INTRODUCTION

This program was required to display radar cross-section (RCS) data collected from field trials allowing the comparison of one plot with another and the processing of the raw data to produce a series of plots of average RCS for various aspect angles. This task therefore involved writing a program that could align and then combine two graphs. The program was structured so that the user could methodically combine the collected data, then produce an average RCS profile for a particular aspect angle of a ship. The data to be processed consisted of amplitude returns from a frequency swept microwave signal centred on the ship being measured. Each data set consists of 255 equally spaced points which represent the relative range in the cross-section profile being looked at and the corresponding amplitude RCS at each range point.

Because the increment between distance measurements is constant and each data set has the same number of data points, combining the graphs could be achieved quite simply. However for more complex data sets with non-linear abscissa values this section of the program would need to be replaced by a more sophisticated process.

When Graph Match is used to combine RCS profiles the user first selects a back chart then combines it with a front chart. The overlying front chart can be moved and rescaled to allow for small movements or changes in foreshortening during a measurement. Finally the two charts may be combined using any specified weighting. The data may then be transformed before plotting so that actual units (dBm2) are represented on the axes, or transformed from a logarithmic scale to a linear scale. Data may also be transformed back to its original units prior to saving the processed RCS profile.

The Apple Macintosh computer was chosen because of its "user friendly" operating system and excellent graphics. Also for the Macintosh it is easy to write interactive graphic programs due to an extensive "tool box" of routines incorporated in the operating system. Graph Match runs on either a Macintosh SE or Macintosh II, and makes full use of colour when it is available. The Macintosh guidelines for writing applications have been diligently followed hence this Macintosh program looks similar to other Macintosh applications.

2 GRAPH MATCH PROGRAM

2.2 Input data specification

A graph or chart is stored as a composite file where the first 4 lines contain header information and subsequent lines contain coordinate (x,y) pairs. Two types of files may be input to the program: either raw data files or files already created by the program.

In the case of raw data, the first line contains the file name. The next 3 lines contain details concerning the file. The coordinate pairs are in integer form; the x-value ranging from 0 to 255 and the y-value ranging from 0 to 400. Typically these files contain 256 coordinate pairs.

In the case of program-generated files, the first line contains the file name. However, the next 3 header lines differ in content from raw data files. For a combined chart, line 2 contains the name of the file used for the back chart and the weighting value used and line 3 contains the name of the file used for the foreground chart and its weighting value (These details can be viewed using Graph Details found in the Screen menu). The subsequent coordinate pairs will be real values.

Currently the program is setup to combine files with 256 coordinate pairs although a file with a different number is accepted.

A file may contain up to 1000 coordinate pairs. Such a file can be successfully displayed and have its values altered. However, when files are to be combined the x-values in both files after all mathematical adjustments have taken place must be in the range -100 to 400 for the combining process to occur successfully.

2.3 Program description

2.3.1 What the display shows

Graph Match has been written to display data sets as graphs allowing their direct visual comparison. The graphs can be displayed with a grid and with or without a numbered axis. The front graph may be moved and its size changed with respect to the back graph; viewing of this rescaling and moving of the front graph being the main function of the display. When the user is satisfied that the two graphs are visually matched, they can then be combined to form an average graph which is displayed along with the front and back graphs from which it was evaluated.

Graphs may be displayed overlaid one over the other, or split into three windows where the hard black-lined top graph is the back chart and the thinner-lined lower graph is the front chart. A combined chart will be overlaid on or displayed between the charts. A front, back or combined chart or any combination thereof may be displayed by selecting from items found in the Screen menu. When the item has a tick alongside it, that indicates it is turned on.

As is typical with a Macintosh application the display also shows pop-up dialogue boxes and pull-down menus for user input and program control. For users of MacIIs with colour displays there is a menu called Colour which allows the three different charts to be displayed in contrasting colours.

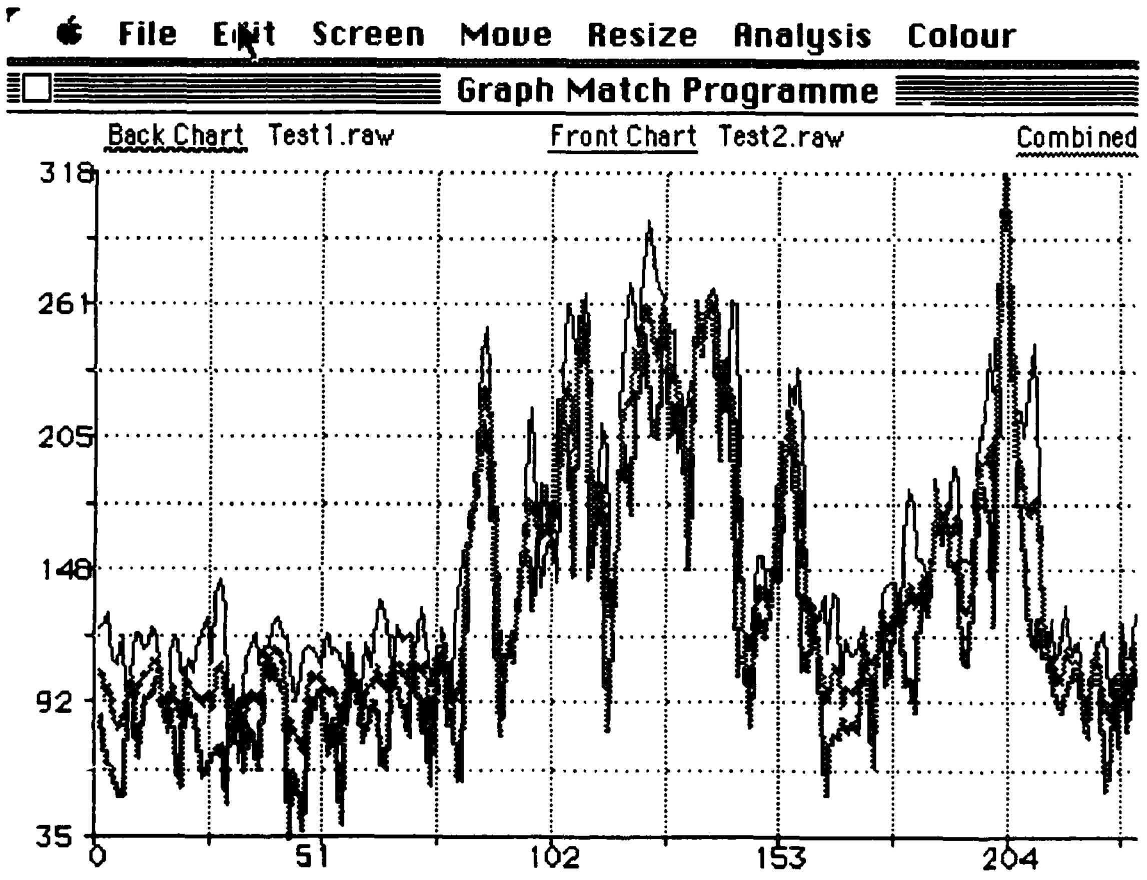


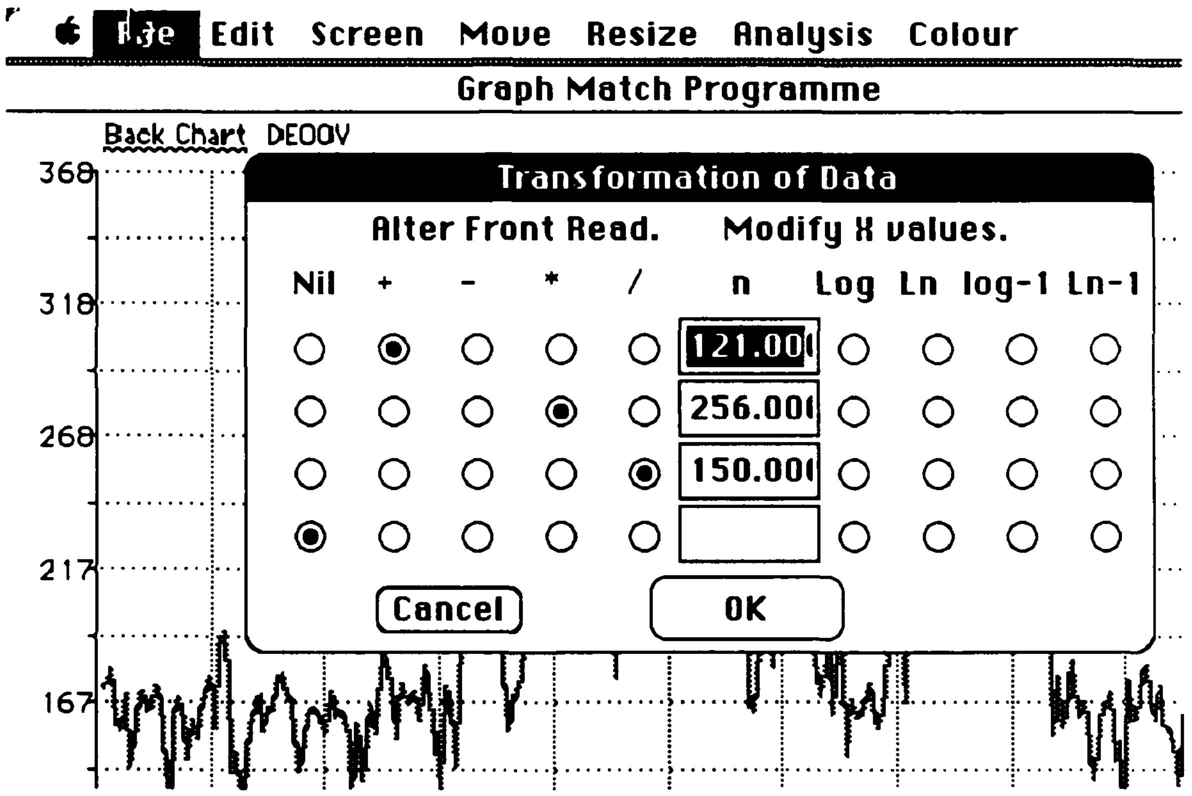
Figure 1 Sample screen

2.3.2 How to use the program

Loading data

When data is loaded into memory it can be read in directly or transformed by specifying the required arithmetic operations. In the first case the **File** menu is pulled down and the back chart chosen. Next the front chart is chosen. This chart may then be repositioned or rescaled.

If a transformation is required, the arithmetic operations are specified by selecting **Set Back Transformation** and/or **Set Front Transformation** from the **File** menu and then filling in the dialogue box(es). Back chart and front chart are then chosen in the normal manner.



Transformation of Data

Alter Front Read.					Modify X values.				
Nil	+	-	*	/	n	Log	Ln	log-1	Ln-1
○	●	○	○	○	121.00	○	○	○	○
○	○	○	●	○	256.00	○	○	○	○
○	○	○	○	●	150.00	○	○	○	○
●	○	○	○	○		○	○	○	○

Figure 2 Transformation dialogue box

Altering graph size and position

The front chart may be moved **left, right**, and **up** or **down** by selecting from the **Move** menu, made **shorter, taller, narrower** or **wider** using the **Resize** menu until the user is satisfied that it is matched with the back chart. The magnitude of the changes may be selected to be 1, 2, 5 or 10 percent of the back chart ranges.

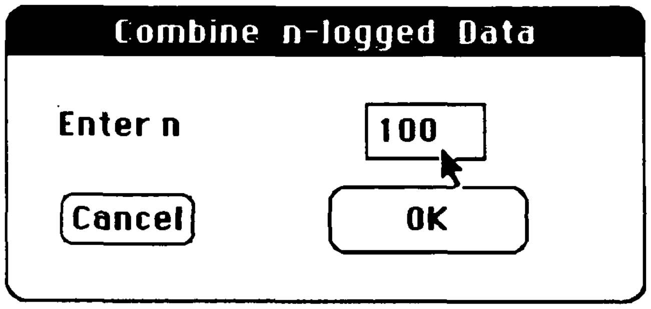
As an alternative to moving the front chart left or right the data may be wrapped around, which involves taking a part of the graph from one end and transforming it to the other end. This effectively repositions the peaks and troughs of the front chart. The **wrapleft** and **wrapright** functions move the data by 1, 2, 5 or 10 data items to the left or right, and are found in the **Move** menu.

If the user wishes to return an altered front chart to its original position and size this can be done using the **Resize to Screen** command found in the **Resize** menu. However, this will not work where the chart has been wrapped around.

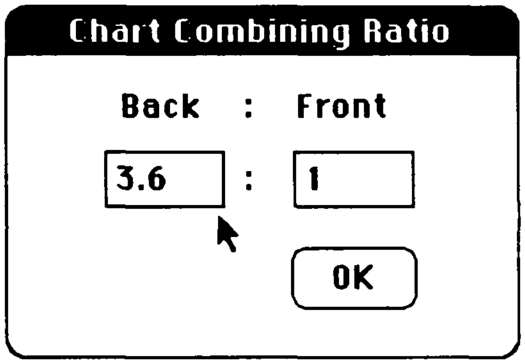
Combining graphs

When the optimum match between the front and back charts has been found, the charts can then be weighted and combined. The charts y-values may be averaged for corresponding x-values. The number of data items in the combined chart that is created corresponds to the number of data items in the back chart present. Alternatively, since they may be on a logarithmic scale before combining, the y-values may be divided by a constant, converted to a linear scale, then added. The process is then reversed to display the combined graph in the original logarithmic scale. **Combine** and **nLog** commands are found in the **Analysis** menu.

Any weighting may be specified with the **Combining Ratio** command.



Enter n: 100



Back : Front
3.6 : 1

Figure 3 Combining Ratio Dialogue Boxes

Printing graphs

For the purpose of taking a hard copy of the computer screen there is a **Print** command found in the **File** menu. When this is invoked, whatever is on the screen will be sent to the printer. What is displayed is determined by the settings in the **Screen** menu, or if two charts have just been combined only the combined chart will be displayed. If the **Graph Details** (header information) are displayed the print item is made grey and cannot be selected. To print the graph(s) the page setup is specified using the dialogue box to select orientation, quality and quantity of prints required.

Saving data

Saving data is similar to loading data. Firstly, standard Macintosh system Input/Output dialogue boxes are used, secondly data may be saved as displayed, or transformed prior to saving. **Set Front Transformation** or **Set Comb Transformation** (found in the **File** menu) can be selected, then the **Save Foreground** or **Save Combination** commands (found in the **File** menu) entered. The only apparent difference to the user between saving and loading data is that when saving data the user must specify a new filename rather than choose from a list of them.

Saved data sets have four header information lines which means they can be reloaded by Graph Match as new charts.

An overview of the sequence of steps involved in using Graph Match follows:

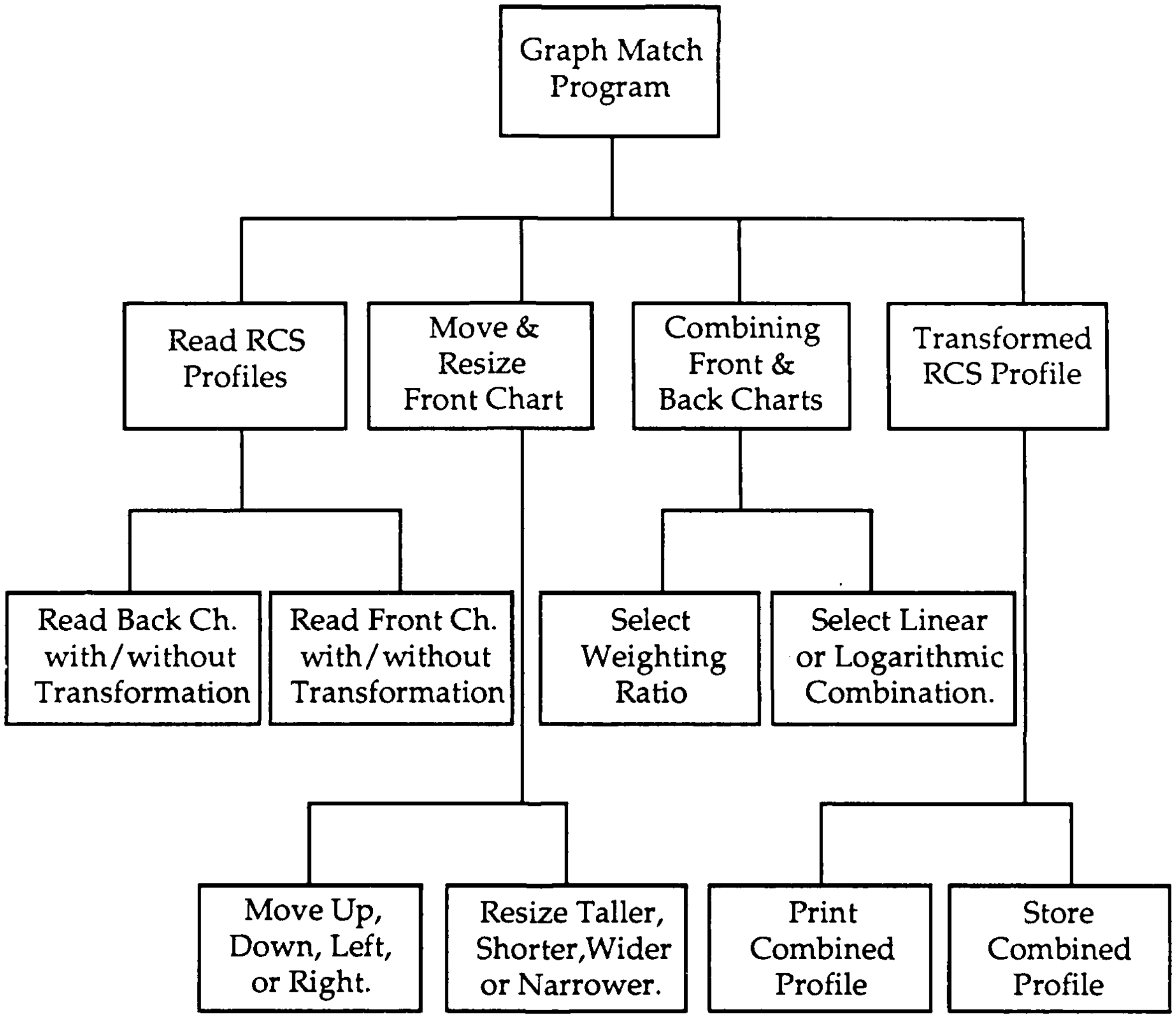

Graph Match Program
Read RCS Profiles
Move & Resize Front Chart
Combining Front & Back Charts
Transformed RCS Profile
Read Back Ch. with/without Transformation
Read Front Ch. with/without Transformation
Select Weighting Ratio
Select Linear or Logarithmic Combination.
Move Up, Down, Left, or Right.
Resize Taller, Shorter,Wider or Narrower.
Print Combined Profile
Store Combined Profile

Figure 4 Functional Chart

2.3.3 Programmer's guide

The program was written using Borland's Turbo Pascal Version 1.0 (Borland 1986) to edit and compile the main source code. Menu details, window specification, dialogue box and item declarations and other required Mac program items were created then modified using Apple Computer's Resedit program.

Throughout the program many of the variable names are self-explanatory. Extensive use is made of global variables .

Except for transformation details, all of the data necessary to manipulate, display, and save the three different charts is kept in a chart data record type. The three globally declared variables which use this record type are back, front and combo.

Currently dialogue box information which is kept in the resource file can be purged. Thus if the program disk is not in the computer when a dialogue box is required, it must be inserted before the dialogue box will appear.

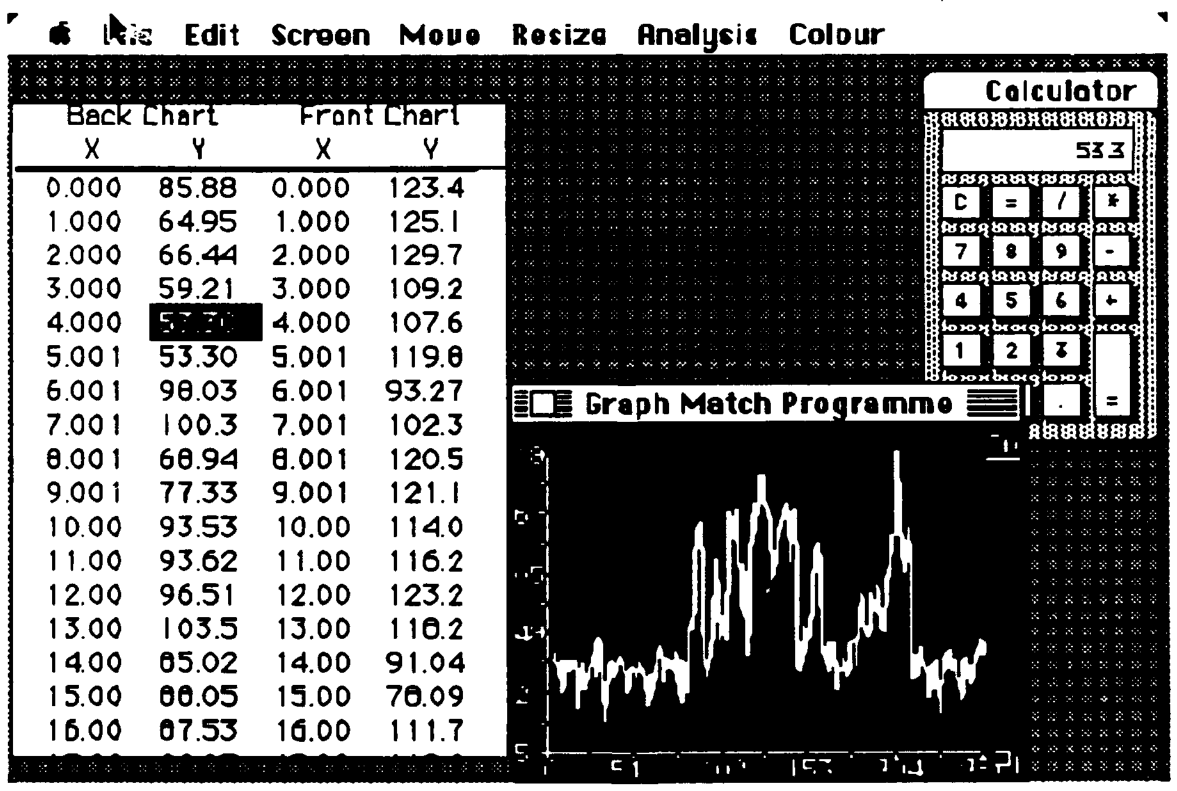
3 DISCUSSION

3.1 Suggestions

First priority should be to improve the basic shell to facilitate automatic event driven window updating. The shell used was taken from another Mac program. Currently this shell does not correctly cater for window updates by responding to update events in the normal manner. Consequently the procedure *Redoscreen* handles the screen update requirements necessary with dialogue boxes and general manipulation of data on the screen.

This program could be further developed to make it more versatile and general in its applications. One area that needs development is the method of combining data when the number of data values of the front and back charts are different. Other useful developments would be to make dialogue box information unpurgeable, and to make the currently unavailable items in the Edit menu fully operational, especially the Data display function which has been partially developed but not incorporated in the current release version of Graph Match (refer to figure 5).

Further development of the combination feature would require careful attention to the mathematics used.



Back Chart		Front Chart	
X	Y	X	Y
0.000	85.88	0.000	123.4
1.000	64.95	1.000	125.1
2.000	66.44	2.000	129.7
3.000	59.21	3.000	109.2
4.000	57.30	4.000	107.6
5.001	53.30	5.001	119.8
6.001	98.03	6.001	93.27
7.001	100.3	7.001	102.3
8.001	68.94	8.001	120.5
9.001	77.33	9.001	121.1
10.00	93.53	10.00	114.0
11.00	93.62	11.00	116.2
12.00	96.51	12.00	123.2
13.00	103.5	13.00	118.2
14.00	85.02	14.00	91.04
15.00	88.05	15.00	78.09
16.00	87.53	16.00	111.7

Figure 5 Data Display

3.2 Limitations

Some printing commands need altering if the program is to be run on a Macintosh SE. Currently the program is setup to combine files with 256 coordinate pairs although a file with a different number is accepted.

3.3 Recommendations

While this program manipulates RCS data adequately it needs further work to make it a more general and versatile program. As we are not aware of any commercial software that can perform the types of functions outlined, Graph Match provides a basis for further development of its software as required.

4 ACKNOWLEDGEMENT

I wish to record the invaluable guidance of Mr Greg Chase, formerly of Electronic Countermeasures Group, for providing a detailed layout of the program design, a working shell from which to start, and for his ongoing support during the program development period.

REFERENCES

Apple Computer Inc, 1988, *Inside Macintosh*, Addison-Wesley, U.S.A.

Borland International, Inc,1986, *Turbo Pascal for the Mac (User's Guide and Reference Manual*, Borland International, California.

APPENDIX I

PROGRAM SOURCE CODE LISTING

```
program GraphMatch; { Author Mar Barrington 7-Dec-1988 }

{$R+}                   { Turn on range checking              }
{$I+}                   { Turn on I/O error checking          }
{$B+}                   { Set bundle bit (for icon, etc.)     }
{$R Gr_Mt.Rsrc}         { Identify resource file              }
{$T APPLPenn}           { Set application ID                  }
{$U-}                   { Turn off auto link to runtime units }
{$S+}

uses PasInOut,SANE,Pasprinter,Memtypes,QuickDraw,OSIntf,ToolIntf,PackIntf,
MacPrint;

const
  MenuCnt        =    8;    { total # of menus                     }
  ApplMenu       = 1000;    { resource ID of Apple Menu            }
  FileMenu       = 1001;    { resource ID of File Menu             }
  EditMenu       = 1002;    { resource ID of Edit Menu             }
  screenMenu     = 1003;    { resource ID of Screen control Menu }
  MoveMenu       = 1004;    { resource ID of Move Menu             }
  ResizeMenu     = 1005;    { resource ID of Resize Menu           }
  AnalysisMenu   = 1006;    { resource ID of Analysis menu         }
  ColourMenu     = 1007;    { resource ID of Colour Choice Menu  }
  AppleM         =    1;    { index into MenuList for Apple Menu }
  FileM          =    2;    { index into menulist for File Menu  }
  EditM          =    3;    { ditto for Edit Menu                  }
  screenM        =    4;    { ditto for Screen Menu                }
  MoveM          =    5;    { ditto for Move Menu                  }
  ResizeM        =    6;    { ditto for Resize Menu                }
  AnalysisM      =    7;    { ditto for Analysis menu              }
  ColourM        =    8;    { ditto for Colour Menu                }
  MainID         = 1000;    { resource ID for MainWindow           }
  DatWID         = 1001;    { resource ID for data Window          }
  nLogdlgID      = 1001;    { resource ID for dialog box           }
  trnsfrm_dlgID  = 1002;    { resource ID for dialog box           }
  Yes_No_dlgID   = 1003;    { resource ID for dialog box           }
  NegNumdlgID    = 1004;    { resource ID for dialog box           }
  ComRatiodlgID  = 1005;    { resource ID for dialog box           }
  SaveDialoguID  = 1006;    { resource Id for dialog box           }
  AboutID        = 1000;    { resource ID for dialog box           }
  {    Title  About Graph Match                                    }
  {    ^0                                                          }
  {    ^1                                                          }
  {    ^2          #1  OK                                          }
  Text1ID        = 1000;    { resource IDs for 'About...' text    }
  Text2ID        = 1001;
  Text3ID        = 1002;
  SaveAlert      = 1001;    { resource ID of save file alert      }
    {  Title                                                       }
    {  ^0                                                          }
    {  #1 Yes       #2 No       #3 Cancel                          }
   OKagainalert  = 1009;    {   ^0    ^1    OK      Try again     }
  myCursID       = 1000;    { resource ID for myCursor             }
  myCursor       =    5;    { array index for myCursor             }
  BSize          =  640;    { buffer size  for Disk I/O prev 512 }
  BCount         =  480;    { buffer count for Disk I/O prev 256 }
  put_up         = true;
  remove        = false;
  maxplot       =  1000;
  useEXscreen   =  0.85;
  useWYscreen   =  0.85;

type

  chart_data_type = record
         EX, WY : array[1..maxplot] of real ; {The Data}
   is_read, set_to_display, showing: boolean;             penpat: pattern;
         last_point, colour, line_thickness : integer;
         EXmin, EXmax, WYmin, WYmax, EXrange, WYrange,
         EXoffset, WYoffset,
         EXoffscale, WYoffscale : real ;
           line_zero,line_one,line_two,
         line_three : string[80] ;
                   end ;

  Pchart_data_type = ^chart_data_type;

  transtype   = ( tnone,tadd,tminus,ttimes,tdiv,tlog,tln,talog,taln );
  transform_type = array[1..4]of
                    record
                      ttype : transtype ;
                            : real ;
                    end ;

  CursorList    = array[iBeamCursor..myCursor] of CursHandle;
  PtrInteger    = ^integer;
  PtrString     = ^str255;
  string80      = string[80];
  string255     = string[255];

var    { graph match vars follow }

  fileselected,    { Used in Read_chart_file                            }
  ok,  info_up,    { Graph details displayed, other items not avail.    }
  grid_up,         { grid is required, and redone every clearwindow     }
  axis_up,         { axis are added when this global boolean is true    }
  overlaid,        { front, back and combo plotted over one another     }
  wiped,           { true if during move or resize plot is not up       }
  comb_just_put_up,{ determines display immediately after combining     }
  normal_comb,     { the front and back records are combined            }
  nlog_comb,       { logged front and back records combined             }
  this_front_saved,{ facilitates tick,indicates graph is saved          }
  this_combo_saved,{ facilitates tick,indicates graph is saved          }
  MacII,           { Used in setting up printing details                }
  sepia,           { if this flagged backpat is 75% white,25% yellow    }
  list_up,         { if data set editing occurring then menus greyed    }
  Finished : boolean;{ used to terminate the program                    }
  pfilename:string80;{ used in Read_chart_file                          }
  infile    : Text ; { used in Read_chart_file                          }
  front_EX_trans,    { details for transformation of front x data.      }
  front_WY_trans,    { details for transformation of front y data.      }
  back_EX_trans,     { details for transformation of front x data.      }
  back_WY_trans,     { details for transformation of front y data.      }
  save_EX_trans,     { details for transformation of front x data.      }
  save_WY_trans,     { details for transformation of front y data.      }
  combo_EX_trans,    { details for transformation of combo x data.      }
  combo_WY_trans: transform_type;{transformation of combined y data.    }
  back,              { charts are Globally decl. chart storage array    }
  front,             { charts are Globally decl. chart storage array    }
  combo: ^chart_data_type;{charts Globally decl. chart storage array    }
  window_EX_scale,    { used as general scaling const changed by grow }
  window_WY_scale,    { used as general scaling const changed by grow }
  prev_EX_scale,  { Grow-compare with standard to rescale move const }
  prev_WY_scale,  { Grow-compare with standard to rescale move const }
  back_weight,        { ratio for weighting when combining            }
  front_weight :real;{ ratio for weighting when combining             }
  cross_point,        { used to store cross location for erasing.     }
  zero_point: point ;{ zero_point is used to set points to 0,0        }
  percent_change,     { How much to move or resize front graph        }
  liftback,           { if split screen, move the back up to the top  }
  EXwindowdots,       { The number of dots available in the window    }
  WYwindowdots,       { Dots are altered during Grow                  }
  prt_x_mv_pixel,     { add to reposition window for laserprinter.    }
  prt_y_mv_pixel,     { add to reposition window for laserprinter.    }
  printEXoffset,      { set to 80 in Init, dependant on SE/MacII.     }
  printWYoffset,      { set to 80 in Init, dependant on SE/MacII.     }
  scale_option,       { Set by Menu  method of make_scale             }
  grid_colour,        { Colour of the grid must contrast to back gind }
  cross_colour,       { colour for marker cross & displayed values.   }
  x_y_values_col,     { & matching displayed values.                  }
  graphdetails_col,   { Graph Details writing colour                  }
  axis_colour,        { Colour of the axis                            }
  numbers_colour,     { Colour of the axis numbers.                   }
  background_colour,  { Colour of the background                      }
  foreground_colour,  { Colour of the foreground pencolour            }
  the_colour_choice : integer ; { colour of srceen, grid and plots    }
  any_changes_to_file :boolean; { may be used, but is a leftover.     }
  Ticks          : Longint;     { keeps track of time                 }
  TV,TH          : Integer;     { location of text                    }
  theEvent       : EventRecord; { event passed from operating system  }
  { Screen stuff }
  DragArea     : Rect;    { defines area where window can be dragged }
  GrowArea     : Rect;    { defines area to which windows size change}
  ScreenArea     : Rect;  { defines screen dimensions                 }
  CursList       : CursorList;  { used to hold cursor handles         }
  { Menu stuff}
  MenuList        : array[1..MenuCnt] of MenuHandle; {holds menu info}
  { Window stuff }
  DatWPtr,                             { pointer to data view window   }
  MainPtr        : WindowPtr;         { pointer to main window        }
  DatWRec,                             { window for data viewing       }
  MainRec        : WindowRecord;      { holds data for main window    }
  MainPeek       : WindowPeek;         { pointer to MainRec            }
  ScreenPort     : GrafPtr;            { pointer to entire screen      }
  FrontWindow    : WindowPtr;          { pointer to active window      }
  { program specific stuff}
  VFlag          : boolean;
  Reply          : SFReply;
  SPtr           : StringPtr;
  test_string    : str255 ;

procedure ClearWindow(WPtr : WindowPtr);
{ purpose          clears window, draws grow stuff, sets location }
var
  TRect        : Rect;
begin
  if (WPtr = MainPtr) and (Wptr = FrontWindow ) then
    begin
      backcolor(background_colour) ;
      forecolor(yellowcolor);
      if sepia then  backpat(ltgray) else backpat(white);
      EraseRect(WPtr^.portRect);       { clear rect area of window    }
      back^.showing := false ;
      front^.showing := false ;
      combc^.showing := false ;
      pensize(0,0) ;
      DrawGrowIcon(WPtr);              { draw grow icon }
      TH := 5; TV := 12                { set text loc to upper left   }
  end
end; { of proc ClearWindow }

procedure WipeWindow(WPtr : WindowPtr);
{  purpose       clears window, draws grow stuff, sets location }
var
  TRect        : Rect;
begin
  if (WPtr = MainPtr) and (Wptr = FrontWindow ) then
    begin
      backcolor(background_colour) ;
      EraseRect(WPtr^.portRect);       { clear rect area of window    }
      pensize(0,0) ;
      DrawGrowIcon(WPtr);              { draw grow icon }
      TH := 5; TV := 12                { set text loc to upper left   }
  end
end; { WipeWindow }

procedure DrawStart(Str : String);{ move to text location, write Str }
begin
  MoveTo(TH,TV);                    { move to current text location  }
  DrawString(Str);                  { write on screen                }
  TH := TH + StringWidth(Str)       { advance location to end of str }
end; { of proc DrawStart }

procedure RealToStr(Val : Real; var Str : String);
var    { purpose         does conversion from real number to string }
  num_format    : decform ;
begin
  num_format.style := fixeddecimal ;
  num_format.digits := 3 ;{ decimal points }
  num2str(num_format,val,Str) ;
end; { of proc RealToStr }

function Int_to_str(i,size:integer):string80;
var s:string;
begin
   numtostring(i,s);
   while length(s) < size do s:=concat(' ',s);
   int_to_str:=s;
end;{Int_to_str}

procedure str_to_int(s:str255;var num:integer;var ok:boolean);
{Convert a string to an integer }
var i:integer;
    n:longint;
begin
  i:=1;
  while i<=length(s) do
    begin
      if s[i]=' ' then delete(s,i,1) else i:=i+1;
    end;
  ok := length(s) > 0;
  if ok then for i:=1 to length(s) do
     if not (s[i] in ['0'..'9']) then ok := false;
  if ok then stringtonum(s,n);
  if ok then num:=n else num:=0;
end;{str_to_int}
```

```
  procedure number_stamp( number:extended; places,hpos,vpos:integer);
  var
    number_string : str255 ;
    num_format    : decform ;
    number_rect   : rect ;
    store_colour  : integer ;
  begin { number_stamp }
   (* store_colour := fore*)
   (* setrect(number_rect,hpos,vpos-9,hpos+40,vpos) ;
    eraserect(number_rect) ;*)
    textsize(11) ;
    MoveTo(hpos,vpos);
    num_format.style := fixeddecimal ;
    num_format.digits := places ;{ decimal points }
    num2str(num_format,number,number_string) ;{ forecolor(stamp_colour) ;}
    PenPat(black) ;
    DrawString(number_string) ;
  end ; { number_stamp }

(*$S Inits *)

function rdreal(s:string80; var position:integer;var ok:boolean):real;
var
  len, i, neg : integer ;
  num, fnum, fact : real ;
  t : string80 ; { t is a working copy of the input string s }
begin { rdreal }
  len  := length(s) ;
  t    := concat(s,' ') ; { Add a blank to make while loop easier }
  ok   := false ;
  num  := 0 ;
  fnum := 0 ;
  fact := 1 ;
  neg  := 1 ;
  if ( position > 0 ) and ( len >= position ) then
    begin
      i := position ;
      while ( t[i] in [' '] ) and ( i < len ) do i := i + 1 ;
           { Skip blanks and commas }
      if t[i] = '-' then
        begin
          neg := -1 ;
          i := i + 1 ;
        end ;
      if i <= len then
        begin
          while(t[i] in ['0'..'9']) and (i<= len) do
            begin
              num := num * 10 + ord(t[i]) - ord('0') ;
                { Build whole part of number }
              i := i + 1 ;
              ok := true ;
            end ;
          if ( i <= len ) and ( t[i] = '.' ) then
            begin  { fraction part }
              i := i + 1 ;
              while ( t[i] in ['0'..'9'] ) and ( i <= len ) do
                begin
                  fact := fact * 0.1 ;
                  fnum := fnum + fact * ( ord(t[i]) - ord('0') ) ;
                    { Build fraction }
                  i := i + 1 ;
                  ok := true ;
                end ;
            end ;
          if ( i <= len ) then ok := ok and ( t[i] in [' ','-'] ) ;
 { Number must be terminated by end of line, space, minus, or comma.}
          position := i ; { Start of next num and end of this number.}
        end ;
    end ;
  if ok then rdreal := neg * ( num + fnum ) else rdreal := 0 ;
end ; { rdreal }

procedure top_of_screen_details;{on screen info about graph}
begin{top_of_screen_details}
  if back^.is_read and back^.set_to_display and not(comb_just_put_up)
  then
    begin{put up back graphs name}
      forecolor(back^.colour) ;{ example line under back chart text }
      penpat(back^.penpat);
  pensize(back^.line_thickness,back^.line_thickness);
      moveto( 39 + prt_x_mv_pixel,13 + prt_y_mv_pixel );
  lineto( 92 + prt_x_mv_pixel,13 + prt_y_mv_pixel );
      pensize(1,1); textsize(10); penpat(black); { back details text }
      moveto( 40 + prt_x_mv_pixel,12 + prt_y_mv_pixel ) ;
  drawstring( 'Back Chart' );
      moveto( 102 + prt_x_mv_pixel,12 + prt_y_mv_pixel ) ;
      penpat(black);
      drawstring( back^.line_zero ) ;
    end; {put up back graphs name}
 if front^.is_read and front^.set_to_display and not(comb_just_put_up)
      then
    begin{put up front graphs name}
      forecolor(front^.colour) ;{example line under front chart text }
      penpat(front^.penpat);
  pensize(front^.line_thickness,front^.line_thickness);
      moveto( 209 + prt_x_mv_pixel,14 + prt_y_mv_pixel );
  lineto( 266 + prt_x_mv_pixel,14 + prt_y_mv_pixel );
      pensize(1,1); textsize(10) ;penpat(black);{ front details text }
      moveto( 210 + prt_x_mv_pixel,12 + prt_y_mv_pixel ) ;
  drawstring( 'Front Chart' ) ;
      moveto( 275 + prt_x_mv_pixel,12 + prt_y_mv_pixel ) ;
      penpat(black);
      drawstring( frcnt^.line_zero ) ;
    end; {put up front graphs name}
  if(combo^.set_to_display)or(comb_just_put_up) then
    begin{put up combo graph indicator}
      forecolor(combo^.colour);
  pensize(combo^.line_thickness,combo^.line_thickness);
  penpat(combo^.penpat);
      moveto( 389 + prt_x_mv_pixel,14 + prt_y_mv_pixel );
  lineto( 466 + prt_x_mv_pixel,14 + prt_y_mv_pixel );
      textsize(10); pensize(1,1);
  penpat(black);
      moveto( 390 + prt_x_mv_pixel,12 + prt_y_mv_pixel ) ;
      penpat(black);
      drawstring( 'Combined Chart') ;
    end;{put up combo graph indicator}
  forecolor(foreground_colour) ;
end; {top_of_screen_details}

procedure centimetre_grid ;
var  line , temp_col, more_lines : integer ;
begin { grid }
  setcursor(curslist[watchcursor]^^);
```

```
    if grid_up then
      begin
        if MacII then more_lines := 4
        else          more_lines := 0 ;
        forecolor(grid_colour) ;
        penpat(ltgray) ;  (*penpat(black);*)
        pensize(1,1) ;
        for line := 0 to (11 + more_lines) do { horiz grid lines in cm }
          begin   {re below: 28.3464is a better conversion }
            moveto(prt_x_mv_pixel, line * 28 + prt_y_mv_pixel );
             lineto(prt_x_mv_pixel+EXwindowdots,
         round(line*28)+prt_y_mv_pixel);
          end ;
        for line := 0 to (18 + more_lines) do {vertical grid lines, cm }
          begin
            moveto( line * 28 + prt_x_mv_pixel, prt_y_mv_pixel ) ;
            lineto( line*28+prt_x_mv_pixel,WYwindowdots+prt_y_mv_pixel);
          end ;
      end ;
    pensize(0,0) ;
  end ; { centimetre_grid }

  procedure grid_and_axis ;
  var
    EXgrid_penpat, WYgrid_penpat: pattern;
    axis_ORIG_EX, axis_ORIG_WY, axis_TOP_WY, axis_RIGHT_EX,
    i, line_colour, numb_colour, EX_dec_places, WY_dec_places: integer ;
    axis_incr_EX, axis_incr_WY, figs_incr_EX, figs_incr_WY,
    EX_bias, WY_bias : REAL ;
  begin { grid_and_axis }
    if(back^.is_read)and(overlaid) then
      begin{(back^.is_read)}
        if axis_up then
          begin{put axis up}
            line_colour := axis_colour ;
            numb_colour := numbers_colour ;
            WYgrid_penpat := ltgray ;
            EXgrid_penpat := gray ;
          end  {put axis up}
        else
          begin{erase axis}
            line_colour := background_colour ;
            numb_colour := background_colour ;
            WYgrid_penpat := black ;
            EXgrid_penpat := black ;
          end ;{erase axis}
        {  prepare screen scales and offsets as per procedure plot }
        EX_bias := prt_x_mv_pixel + EXwindowdots*((1.0-useEXscreen)/2 );
       { Start 5% in from the left, & Start 5% down from the top }
        WY_bias := prt_y_mv_pixel +
           (WYwindowdots*(useWYscreen+(1.0-useWYscreen)/2)) - liftback;
       {calculate actual origin and maxscale screen pos and increments}
        axis_ORIG_EX := Round(EX_bias)-2;
        axis_ORIG_WY := Round(WY_bias);
        axis_RIGHT_EX:= Round(EX_bias+(back^.EXrange*window_EX_scale));
        axis_TOP_WY  := Round(WY_bias-(back^.WYrange*window_WY_scale));
        axis_incr_EX := (axis_RIGHT_EX-axis_ORIG_EX)/10;{x marker inc }
        axis_incr_WY := (axis_TOP_WY - axis_ORIG_WY)/10;{y marker inc}
        figs_incr_EX := back^.EXrange / 5 ;  {x number increment}
        figs_incr_WY := back^.WYrange / 5 ;  {y number increment}
        {draw axis lines}
        penpat(black) ; pensize(1,1) ; forecolor(line_colour) ;
        moveto( axis_ORIG_EX, axis_TOP_WY  );
      lineto( axis_ORIG_EX, axis_ORIG_WY );
      lineto( axis_RIGHT_EX,axis_ORIG_WY );
      {dependent on data, set decimal places for axis numbers}
      forecolor(numb_colour) ;
      if      back^.EXmax < 0.1 then EX_dec_places := 4
      else if back^.EXmax < 1   then EX_dec_places := 2
      else if back^.FXmax < 10  then EX_dec_places := 1
      else                           EX_dec_places := 0;
      number_stamp( back^.EXmin, EX_dec_places, axis_ORIG_EX-2,
axis_ORIG_WY+12 );
      if      back^.WYmax < 0.1 then WY_dec_places := 4
      else if back^.WYmax < 1   then WY_dec_places := 2 {min values}
      else if back^.WYmax < 10  then WY_dec_places := 1 {special pos}
      else                           WY_dec_places := 0;
      number_stamp( back^.WYmin, WY_dec_places, axis_ORIG_EX-22,
axis_ORIG_WY+5);
      {loop around 10 times, putting in all x&y markers and numbers}
      for i := 0 to 10 do { 0..8 }
        begin{ put in markers, grid lines, numbers, both scales. 0..8}
          forecolor(line_colour);penpat(black); {x mark}
  {x mark}moveto(Round(axis_ORIG_EX+i *axis_incr_EX),axis_ORIG_WY+3);
  {x mark}lineto(Round(axis_ORIG_EX+i *axis_incr_EX),axis_ORIG_WY);
          if (grid_up) and (axis_up) and( i <> 0 ) then {x grid}
            begin{x grid}
              forecolor(grid_colour);penpat(EXgrid_penpat);{x grid}
              lineto(Round(axis_ORIG_EX+i*axis_incr_EX),axis_TOP_WY );
        forecolor(grid_colour);penpat(black);
            end;{x grid}
          forecolor(numb_colour);penpat(black); {x numb}
          if ( i <> 0 ) and ( not odd(i) ) then {x numb}
            number_stamp( back^.EXmin+i/2*figs_incr_EX, EX_dec_places,
       {x numb}   Round(axis_ORIG_EX+i*axis_incr_EX-
   10),axis_ORIG_WY+12);
          forecolor(line_colour);penpat(black);{y mark}
 {y mark} moveto( axis_ORIG_EX-3, Round(axis_ORIG_WY+i *
   axis_incr_WY));
              lineto( axis_ORIG_EX,Round(axis_ORIG_WY+i*axis_incr_WY));
             if (grid_up) and (axis_up) and( i <> 0 ) then{y grid}
             begin {y grid}
              forecolor(grid_colour);penpat(WYgrid_penpat);
              lineto(axis_RIGHT_EX,Round(axis_ORIG_WY+i
   *axis_incr_WY));
              forecolor(grid_colour);penpat(black);
             end;{y grid}
          forecolor(numb_colour);penpat(black);{y numb}
          if ( i <> 0 ) and ( not odd(i) ) then{y numb}
              number_stamp(back^.WYmin+i/2*figs_incr_WY,WY_dec_places,
       axis_ORIG_EX-22,Round(axis_ORIG_WY+i*axis_incr_WY+5));
        end ;{ put in markers then numbers x, then y scale , 0..8 }
    end;{back^.is_read}
  if(not axis_up)or(not back^.is_read)or(axis_up and not overlaid)then
      centimetre_grid ;
end ;{ grid_and_axis }


  procedure implement_trans( var DT: real; this_trans: transform_type;
       var  trans_ok: boolean ) ;
  var
    row     : integer;
    temp_DT: real   ;

    procedure call_alert;
    var  itemhit: Integer; NegNumPtr: DialogPtr;
    begin{ call_alert }
      SetCursor(Arrow);
      NegNumPtr := getNewDialog(NegNumdlgID,NIL,Pointer(-1)) ;
      ModalDialog(NIL,itemhit); { put dialog box up, get result }
      if itemhit = 1 then DisposDialog(NegNumPtr);{dismiss dialog box}
      trans_ok := false;
    end ;{ call_alert }

  begin {implement_trans}
    for row := 1 to 4 do
      begin{each row}
        case this_trans[row].ttype of
           tnone : ;
           tadd  : DT := DT + this_trans[row].n ;
           tminus: DT := DT - this_trans[row].n ;
           ttimes: DT := DT * this_trans[row].n ;
           tdiv  : DT := DT / this_trans[row].n ;
           tlog  : begin if DT <= 0.0 then call_alert
                     else temp_DT := ln(DT)/2.30258509299;
          DT := temp_DT;  end;
           tln   : begin if DT <= 0.0 then call_alert
                     else temp_DT := ln(DT); DT := temp_DT; end;
           talog : begin if DT <= 0.0 then call_alert
                     else temp_DT := Xpwry( 10, DT );
           DT := temp_DT; end;
           taln  : begin if DT <= 0.0 then call_alert
                     else temp_DT := exp(DT); DT := temp_DT; end;
        end;{of case}
      end;
  end ; {implement_trans}

procedure Save_Foreground;
var
  topleft        : Point ;
  i              : integer ;
  EX_to_save,
  WY_to_save,
  EX_prior_to_trans,
  WY_prior_to_trans : real ;
  reply          : sfreply ;
  result         : oserr ;
  outfile        : text ;
  EX_str,
  WY_str,
  save_str       : String ;
  trans_ok       : boolean ;
begin{ Save_Foreground }
  setcursor(curslist[watchcursor]^^);  { select the new file name }
  topleft.h := 90;
  topleft.v := 100;
  sfputfile(topleft,'Enter new file name',front^.line_zero+'.frt',
     nil, reply);
  if reply.good then
    begin
      setcursor(curslist[watchcursor]^^);
      pfilename   := reply.fname;
      result:=setvol(nil,reply.vrefnum);  { save file to disk }
      ClearWindow(MainPtr);
      rewrite(outfile,pfilename);
      this_front_saved := true ;
      writeln( outfile, front^.line_one+'     ' ) ;
      writeln( outfile, front^.line_two ) ;
      writeln( outfile, front^.line_three ) ;
      writeln( outfile ) ; { blank line }
      grid_and_axis ;
      trans_ok := true;
      i := 1 ;
      repeat{Note EX[i] and WY[i]array vals unchanged at this point.}
          EX_to_save := (front^.EX[i] - back^.EXmin) *
  front^.EXoffscale + back^.EXmin + front^.EXoffset;
          WY_to_save := (front^.WY[i] - back^.WYmin) *
  front^.WYoffscale + back^.WYmin + front^.WYoffset;
          implement_trans( EX_to_save, save_EX_trans, trans_ok ) ;
          implement_trans( WY_to_save, save_WY_trans, trans_ok ) ;
          RealToStr( EX_to_save, EX_str);
          RealToStr( WY_to_save, WY_str);
          save_str := EX_str + ' ' + WY_str;
          writeln( outfile,save_str );
          i := i + 1;
      until ( i > front^.last_point ) or (not trans_ok); {save data }
      close(outfile) ;
    end;
  FlushEvents(everyEvent,0);    { clear events from previous state  }
end;{ Save_Foreground }

procedure Save_Combination ;
var
  topleft         : Point ;
  i               : integer ;
  modified_EX,
  modified_WY     : real ;
  reply           : sfreply ;
  result          : oserr ;
  outfile         : text ;
  EX_str, WY_str,
  type_combine_str,
  save_str        : String ;
  trans_ok        : boolean ;
begin{ Save_Combination }
  setcursor(curslist[watchcursor]^^);  { select the new file name }
  topleft.h := 90;
  topleft.v := 100;
  sfputfile(topleft,'Enter new file name',combo^.line_zero+'.com',
      nil, reply);
  if reply.good then
    begin{go ahead and save combination}
      setcursor(curslist[watchcursor]^^);
      pfilename   := reply.fname;
      result:=setvol(nil,reply.vrefnum);  { save file to disk }
      ClearWindow(MainPtr);
      rewrite(outfile,pfilename);
      this_combo_saved := true ;
      writeln( outfile, combo^.line_one ) ;
      writeln( outfile, combo^.line_two ) ;
      if nlog_comb then type_combine_str := 'nlog used'
      else type_combine_str := 'combined by adding';
      writeln( outfile,type_combine_str ) ;
      writeln( outfile ) ;
      grid_and_axis ;
      trans_ok := true;
      i := 1 ;
      repeat
          modif'ed_EX := combo^.EX[i] ;
          implement_trans( modified_EX, combo_EX_trans, trans_ok ) ;
          modified_WY := combo^.WY[i] ;
          implement_trans( modified_WY, combo_WY_trans, trans_ok );
          RealToStr( modified_EX, EX_str);
          RealToStr( modified_WY, WY_str);
          save_str := EX_str + ' ' + WY_str ;
          writeln( outfile,save_str ) ;
          i := i + 1 ;
      until ( i > combo^.last_point ) or (not trans_ok);
   close(outfile) ;
     end; {go ahead and save combination}
   FlushEvents(everyEvent,0);     { clear events from previous state  }
end;{ Save_Combination }



(*$S           *)


procedure Read_chart_file(     reference: boolean;
                           var chart    : Pchart_data_type;
                           var read_ok  : boolean          );
var
  topLeft                                    : Point ;
  FileFilter                                 : SFTypeList ;
  theErr                                     : OSErr ;
  string_input, s, AString, msg              : str255 ;
  ok, trans_ok                               : boolean ;
  i, position, last_point                    : integer ;
  this_EX_trans, this_WY_trans               : transform_type;
begin { Read_chart_file }
  msg := 'Chart Read' ;
  topLeft.h := 90; { top left horiz point for Get File dialog }
  topLeft.v := 80; { top left vert     "    "    "    "    "    }
  SFGetFile(topLeft,msg,nil,-1,FileFilter,nil,Reply); { select file }
  if Reply.Good then {ok pressed - proceed with file read process}
    begin { proceed with file read and data process }
      setcursor(curslist[watchcursor]^^);
      FileSelected := true;
      pfilename := reply.fname;
      theErr := SetVol(nil,reply.vrefnum); {set vol name to def vol}
      ClearWindow(MainPtr);
      if reference then (*reference true if back chart being read*)
        begin this_EX_trans := back_EX_trans;
       this_WY_trans := back_WY_trans; end
      else{ use front transformation specification }
        begin this_EX_trans:= front_EX_trans;
       this_WY_trans:= front_WY_trans; end;
      reset(infile,pfilename);
      chart^.line_zero := pfilename ;
      readln( infile, chart^.line_one ) ;
      readln( infile, chart^.line_two ) ;
      readln( infile, chart^.line_three ) ;
      readln( infile) ;
      moveto(50,80);textsize(18); penpat(black) ;
      forecolor(chart^.colour) ;
      drawstring('Data is now being read.');{message while user waits}
      chart^.EXmin := 30000 ; chart^.EXmax := -30000 ;{ init mins }
      chart^.WYmin := 30000 ; chart^.WYmax := -30000 ;{  & max's. }
      i := 0 ;  { initialize counter }
      trans_ok := true;{ initialize  a boolean }
      repeat{ repeat while not eof and trans_ok }
        readln( infile, string_input ) ;{ Read a line  as a string.}
        if length(string_input) >= 3 then{length<3,read past blanks}
          begin{process data}
```

```
            position := 1;{ position for FUNC rdreal STRING to REALS.}
            i := i + 1 ;    { array index. }
            chart^.EX[i]:=rdreal(string_input,position,ok);{Convl str}
              implement_trans( chart^.EX[i], this_EX_trans, trans_ok ) ;
{tEX,WY real}chart^.WY[i] := rdreal( string_input, position,ok);
            implement_trans( chart^.WY[i], this_WY_trans, trans_ok ) ;
{ Minimum and maximum of the horizontal EX data & vertical WY data.}
      if chart^.EX[i] < chart^.EXmin then
       chart^.EXmin := chart^.EX[i];
      if chart^.EX[i] > chart^.EXmax then               chart^.EXmax    :=
chart^.EX[i];
      if chart^.WY[i] < chart^.WYmin then
       chart^.WYmin := chart^.WY[i];
         if chart^.WY[i] > chart^.WYmax then
       chart^.WYmax := chart^.WY[i];
          end;{process data}
      until(EOF(infile))or(not trans_ok);{while not eof&trans_ok }
      close(infile) ;
      chart^.last_point := i ;   { Record the last array index. }
      chart^.EXrange  := chart^.EXmax - chart^.EXmin ;
      chart^.WYrange  := chart^.WYmax - chart^.WYmin ;
      chart^.EXoffset := 0 ; { for moving frontplot in horizontal }
      chart^.WYoffset := 0 ; { for moving frontplot in vertical   }
      chart^.EXoffscale := 1 ;{ for altering frontplot EX scaling }
      chart^.WYoffscale := 1 ;{ for altering frontplot WY scaling }
      if reference then
        begin{ new back chart-set scales }
          if(trans_ok)and(chart^.EXrange<>0)and(chart^.WYrange<>0)then
            begin{ ..but only if the data was good, Use 90% of window}
              window_EX_scale:=useEXscreen*EXwindowdots/back^.EXrange;
              window_WY_scale:=useWYscreen*WYwindowdots/back^.WYrange;
      if not overlaid then window_WY_scale := window_WY_scale/2;
              read_ok := true ; {a successful read}
            end {setting window scales to back chart}
          else {trans_ok may have been true, but data was not good }
            begin{read has failed}
              trans_ok := false ;
              read_ok  := false ;{unsuccessful read, so redo_screen}
            end; {read has failed}
        end; { new back chart-set scales }
      chart^.is_read := trans_ok; { boolean value }
      chart^.set_to_display := trans_ok ;
      normal_comb   := false ;    {front or back chart changed }
      nlog_comb    := false ;
      this_front_saved := false ;
      this_combo_saved := false ; { boolean's for setting ticks}
      comb_just_put_up := false ;
    end { ok pressed - proceed with file read and data process }
  else read_ok := false;          { Cancel button was pushed. }
  FlushEvents(everyEvent,0);  { clear events from previous state }
end;{ Read_chart_file }


procedure plot(viewable:boolean;liftback:integer;var chart:Pchart_data_type);
var              { NOTE Subtract + WY's to proceed up from bottom.}
  i, in_from_left, up_from_bottom, xplot, yplot : integer ;
begin { plot }
  setcursor(curslist[watchcursor]^^); penpat(chart^.penpat);
  chart^.showing := viewable;          { .showing is for Menu Ticks }
  if not viewable then forecolor(background_colour)
  else                  forecolor(chart^.colour);
  in_from_left:=Round(EXwindowdots*( (1.0-useEXscreen)/2 ));{7.5% in}
```

```
    up_from_bottom := Round(WYwindowdots*( useWYscreen+(1.0-
    useWYscreen)/2 ))- liftback;  { PIXELS }
  { Take data, Subtract back data minimum value, Scale it (if front),}
  { add offset (if front), ROUND data ,add x data or subtract y data,}
  { Window scale the lot, and add window pixel position offset to it.}
  { Note prt_x_mv_pixel & prt_y_mv_pixel zero except when printing.  }
    pensize(0,0);
    for i := 1 to chart^.last_point do
      begin{draw line}
        xplot := prt_x_mv_pixel + in_from_left + round( (chart^.EXoffset
    +(chart^.EX[i]-back^.EXmin)*chart^.EXoffscale)*window_EX_scale );
        yplot := prt_y_mv_pixel + up_from_bottom -round((chart^.WYoffset
    +(chart^.WY[i]-back^.WYmin)*chart^.WYoffscale)*window_WY_scale);
        lineto( xplot, yplot );
        if i = 1 then
       pensize(chart^.line_thickness,chart^.line_thickness);
      end; {draw line}
    pensize(1,1) ;
end ; { plot }


procedure redo_screen ;
begin { redo_screen }
    setcursor(curslist[watchcursor]^^);
    ClearWindow(MainPtr);                        { Empty the window.   }
    grid_and_axis ; { Grid is put_up - if grid_up = true, axis - ditto }
    comb_just_put_up := false ;
    if( back^.is_read)and( back^.set_to_display)then
     plot(put_up,liftback,back);
    if(front^.is_read)and(front^.set_to_display)then    plot(put_up,0,front);
    if (combo^.is_read) and (combo^.set_to_display)then
     plot(put_up,round(liftback/2),combo) ;
    top_of_screen_details;  {on screen info re which graphs are read in}
end ; { redo_screen }


procedure do_print;
var prrechdl:thprint;
    myprport:tpprport;
    pg,mypgcount:integer;
    all_printed:boolean;

    procedure do_print_screen ;
    var
        proper_back_thickness: integer;
        proper_back_penpat,
        proper_combo_penpat: pattern;

    begin { do_print_screen }
      setcursor(curslist[watchcursor]^^);
      WipeWindow(MainPtr);{dont cancel set_to_display vars!}
      prt_x_mv_pixel :=  printEXoffset ;
      prt_y_mv_pixel :=  printWYoffset ;
      proper_back_thickness := back^.line_thickness ;
      back^.line_thickness := 1 ;
      proper_back_penpat := back^.penpat;
      back^.penpat := dkgray ;
      proper_combo_penpat := combo^.penpat;
      grid_and_axis;{ Grid is put_up - if grid_up = true,axis -ditto}
      if back^.set_to_display then plot(put_up,liftback,back) ;
      if front^.set_to_display then plot(put_up,0,front) ;
      if(not back^.set_to_display)and(not front^.set_to_display)then
    combo^.penpat := black
```

```
        else combo^.penpat := gray ;{comb plot must be distinguishable}
        if combo^.set_to_display then
    plot(put_up,round(liftback/2),combo) ;
        top_of_screen_details;  {on screen info re graphs names}
        back^.line_thickness := proper_back_thickness;
        back^.penpat         := proper_back_penpat;
        combo^.penpat        := proper_combo_penpat;
        prt_x_mv_pixel := 0 ;
        prt_y_mv_pixel := 0 ;
      end ; { do_print_screen }

begin
  PrOpen;
  prrechdl := THprint(Newhandle(sizeof(Tprint)));
  Printdefault(Prrechdl);
  if prstldialog(prrechdl) then;
  if prjobdialog(prrechdl) then
    begin
      myprport:=propendoc(prrechdl,nil,nil);
      pg:=0;
         if prerror = noerr then
          begin
            pg:=pg+1;
            propenpage(myprport,nil);
            if prerror=noerr then do_print_screen; {put up the charts}
            prclosepage(myprport);
           end;
       prclosedoc(myprport);
    end;
  PrClose;
  SetPort(MainPtr);
end;{do_print}


procedure overlay( required : boolean ) ;
begin { overlay }
  if required then
    begin
      if not overlaid then
        begin { overlay plots }
          window_WY_scale  := window_WY_scale * 2 ;
          liftback          := 0 ;  { used to reposition back chart}
          overlaid          := true ;
        end  ;  { overlay plots }
    end
  else if overlaid then
    begin { split screen }
       liftback := round( WYwindowdots * 0.93 / 2 );{repos back chart}
   window_WY_scale  := window_WY_scale / 2 ;
      overlaid          := false ;
    end ; { split screen }
  redo_screen ;
end ; { overlay }


procedure wrap_left ;
var  i , points_to_move : integer ; temp : array[0..300] of real ;
begin { wrap_left }
  setcursor(curslist[watchcursor]^^);
  front^.set_to_display := true ;
  if not overlaid then plot(remove,0,front) ;
  points_to_move := percent_change;{not a percentage, previously 10}
  for i := 1 to front^.last_point do
```

```
    begin
      temp[i] := front^.WY[i] ; { fill temp array with WY values }
    end ;
  for i := 1 to front^.last_point - points_to_move do
    begin {refill front^.WY[i] with values  moved 1or10 points L or R}
      front^.WY[i] := ROUND( temp[i + points_to_move] );
    end ;
  for i := 1 to points_to_move do
    begin
      front^.WY[i + front^.last_point-points_to_move]:=ROUND(temp[i]);
    end ;
  normal_comb := false ;
  nlog_comb := false ;
  this_front_saved := false ;
  this_combo_saved := false ;
  if overlaid then redo_screen
  else plot(put_up,0,front) ;
end ; { wrap_left }


procedure wrap_right ;
var  i , points_to_move : integer ; temp : array[0..300] of real ;
begin { wrap_right }
  setcursor(curslist[watchcursor]^^);
  front^.set_to_display := true ;
  if not overlaid then plot(remove,0,front) ;
  points_to_move := percent_change ;
  for i := 1 to front^.last_point do
    begin
      temp[i] := front^.WY[i] ; { fill temp array with WY values }
    end ;
  for i := 1 to front^.last_point - points_to_move do
    begin    {refill front^.WY[i] with values moved 10 points Right}
      front^.WY[i + points_to_move] := ROUND( temp[i] ) ;
    end ;
  for i := 1 to points_to_move do
    begin
      front^.WY[i]:=ROUND(temp[i+front^.last_point-points_to_move ]);
    end ;
  normal_comb := false ;
  nlog_comb := false ;
  this_front_saved := false ;
  this_combo_saved := false ;
  if overlaid then redo_screen
  else plot(put_up,0,front) ;
end ; { wrap_right }


procedure combine_charts(  EN : real ) ;
(*type
     comb_type = array[-100..400] of real;*)
var
     i, j, start, finish           : integer ;
     WY_value,  b_weight, f_weight, common_denominator,
     indexEX_realign_factor        : real ;
     scbck, frsav, combWY          : array[-100..400] of real;
     ba_w_str, fr_w_str            : String ;
begin
  setcursor(curslist[watchcursor]^^);
  if overlaid then ClearWindow(MainPtr)    { Empty the window.  }
  else if combo^.is_read and combo^.set_to_display then
      plot(remove,round(liftback/2),combo) ;
  grid_and_axis ;    { Grid & Axis put_up }
```

```
setcursor(curslist[watchcursor]^^);
for j := -100 to 400 do
  begin { init arrays }
    frsav[j] := 0.0 ;
    scbck[j] := 0.0 ;
  end ; { init arrays }
indexEX_realign_factor:=(back^.last_point-1)/(back^.EXmax -
       back^.EXmin);
for i := 1 to back^.last_point do
  begin
    j :=  round((back^.EX[i]-back^.EXmin)*indexEX_realign_factor);
    WY_value :=  ( back^.WY[i] - back^.WYmin );
    scbck[j] := WY_value ;
    scbck[j+1] := WY_value ;
  end;{ writing back chart save type data to the mono element array}
for i := 1 to back^.last_point do
  begin
    j :=  round(   ( (front^.EX[i] - back^.EXmin)*front^.EXoffscale
       +front^.EXoffset ) * indexEX_realign_factor );
    WY_value := ( front^.WYoffset/window_WY_scale +
                ( front^.WY[i] - back^.WYmin ) * front^.WYoffscale);
    frsav[j] := WY_value ;
    frsav[j+1] := WY_value ;
  end ; { writing front chart save type data to mono element array }
common_denominator := back_weight + front_weight ;
b_weight := back_weight / common_denominator ;{Combine these arrays}
f_weight := front_weight / common_denominator ;
if EN = 0.0 then
  begin{ standard addition of data }
    for j := -100 to 400 do
      begin { combine the two WY-only arrays }
        combWY[j] := (frsav[j] * f_weight) + (scbck[j] * b_weight) ;
      end ;  { combine two WY only arrays }
      normal_comb := true ;
      nlog_comb := false ;
  end  { standard addition of data }
else
  begin{ data antilog'd,added,then log'd }
    for j := -100 to 400 do
      begin
         combWY[j] := EN * ln( Xpwry(10,frsav[j]/EN)*f_weight
     + Xpwry(10,scbck[j]/EN)*b_weight ) / 2.30258509299 ;
      end;
    nlog_comb := true ;
    normal_comb := false ;
  end; { data antilog'd,added,then log'd }
combo^.last_point := back^.last_point ;{Insert into chart_data_type}
for i := 1 to combo^.last_point do
  begin { insert details into a standard chart_data_type array }
     combo^.EX[i] := ( i - 1 + back^.EXmin  )/indexEX_realign_factor;
combo^.WY[i] := combWY[i-1] + back^.WYmin ;
  end ; { insert details into a standard chart_data_type array }
combo^.EXrange :=  0;{These combo elements are all initialized here}
combo^.WYrange :=  0 ;
combo^.EXmin    :=  0 ;
combo^.WYmin     := 0 ;
combo^.EXoffset := 0 ;{could be for moving comb about if necessary}
combo^.WYoffset := 0 ;
combo^.EXoffscale := 1;{& for altering the comboplot scaling used.}
combo^.WYoffscale := 1 ;
combo^.is_read := true ;
if not overlaid then combo^.set_to_display := true ;
if nlog_comb and not overlaid then redo_screen else
```

```
  plot(put_up,round(liftback/2),combo) ;
  comb_just_put_up := true ;
  RealToStr(  back_weight, ba_w_str );
  RealToStr( front_weight, fr_w_str ) ;
  combo^.line_zero := back^.line_zero+front^.line_zero ;
  combo^.line_one := 'Back Used  Was '+back^.line_zero+'    '+
               'Ratio of '+ba_w_str;
  combo^.line_two := 'Front Used  Was '+front^.line_zero+'    '+
             'is to '+fr_w_str+' weighting.';
  top_of_screen_details; {on screen info re which graphs are read in}
  this_combo_saved := false ;
  back_weight := 1.0;
  front_weight := 1.0;
end ;{ combine }


procedure combining_ratio ;
 var
   itemtype, position         : Integer;
   ok_button                  : boolean ;
   ComRatioDialPtr            : DialogPtr;
   theItem                    : Handle ;
   display_rectangle          : Rect ;
   backR_str, frontR_str      : str255 ;
 begin{ combining_ratio }
   ComRatioDialPtr := getNewDialog(ComRatiodlgID,NIL,Pointer(-1)) ;
   ModalDialog(NIL,itemtype); { put dialog box up; get result }
   if itemtype = 1 then
     begin{ ok button pressed}
       GetDItem(ComRatioDialPtr,3,itemtype,theitem,display_rectangle);
       GetIText(theitem,backR_str) ;
       position := 1 ;
       back_weight := rdreal( backR_str, position, ok );{Convert str}
       GetDItem(ComRatioDialPtr,4,itemtype,theitem,display_rectangle);
       GetIText(theitem,frontR_str) ;
       position := 1 ;
       front_weight:= rdreal( frontR_str, position, ok );{Convert str}
     end ;{ ok button pressed}
(* else
     begin{ cancel button pressed }
       back_weight := 1.0;
       front_weight := 1.0;
     end ;{ cancel button pressed }*)
   DisposDialog(ComRatioDialPtr);      { get rid of dialog box      }
   redo_screen ;                       { return to previous screen }
end;{ combining_ratio }


procedure colourize( item : integer ) ;
begin{ colourize }
  setcursor(curslist[watchcursor]^^);
  info_up := false ;{in case graph details were up immediately prior}
  the_colour_choice := item ;
  case Item of
    1 : begin{ colour against blue }
          foreground_colour := whitecolor ;
          background_colour := cyancolor  ;
          grid_colour       := blackcolor ;
          cross_colour      := magentacolor ;
          x_y_values_col    := magentacolor ;
          graphdetails_col  := whitecolor ;
          axis_colour       := whitecolor ;
          numbers_colour    := whitecolor ;
```

```
        back^.colour       := redcolor ;
        front^.colour      := whitecolor   ;
        combo^.colour      := yellowcolor;
        front^.penpat      := black  ;
        back^.penpat       := black ;
        combo^.penpat      := black  ;
        front^.line_thickness := 2 ;
        back^.line_thickness  := 2 ;
        combo^.line_thickness := 2 ;
        sepia := false ;
      end ;{ colour against blue }
   2 : begin{ colour against white }
        foreground_colour := blackcolor ;
        background_colour := whitecolor ;
        grid_colour       := blackcolor ;
        cross_colour      := magentacolor ;
        x_y_values_col    := magentacolor ;
        graphdetails_col  := blackcolor ;
        axis_colour       := blackcolor ;
        numbers_colour    := blackcolor ;
        back^.colour       := redcolor;
        front^.colour      := bluecolor  ;
        combo^.colour      := greencolor ;
        front^.penpat      := black ;
        back^.penpat       := black ;
        combo^.penpat      := black ;
        front^.line_thickness := 1 ;
        back^.line_thickness  := 1 ;
        combo^.line_thickness := 1 ;
        sepia := false ;
      end ;{ colour against white }
    3 : begin{ colour against white }
        foreground_colour := blackcolor ;
        background_colour := whitecolor ;
        grid_colour       := blackcolor ;
        cross_colour      := magentacolor ;
        x_y_values_col    := magentacolor ;
        graphdetails_col  := blackcolor ;
        axis_colour       := blackcolor ;
        numbers_colour    := blackcolor ;
        back^.colour       := redcolor;
        front^.colour      := bluecolor  ;
        combo^.colour      := greencolor ;
        front^.penpat      := black ;
        back^.penpat       := black ;
        combo^.penpat      := black ;
        front^.line_thickness := 1 ;
        back^.line_thickness  := 1 ;
        combo^.line_thickness := 1 ;
        sepia := true ;
      end ;{ colour against white }
   4 : begin{ black against white }
        foreground_colour := blackcolor ;
        background_colour := whitecolor ;
        grid_colour       := blackcolor ;
        cross_colour      := whitecolor ;
        x_y_values_col    := blackcolor ;
        graphdetails_col  := blackcolor ;
        axis_colour       := blackcolor ;
        numbers_colour    := blackcolor ;
        back^.colour       := blackcolor ;
        front^.colour      := blackcolor ;
        combo^.colour      := blackcolor ;
```

```
            front^.penpat      := black  ;
            back^.penpat       := dkgray ;
            combo^.penpat      := gray   ;
            front^.line_thickness := 1 ;
            back^.line_thickness  := 2 ;
            combo^.line_thickness := 2 ;
            sepia := false ;
          end ;{ black against white }
      5 : begin{ make white against black monochrome  monitor.}
            foreground_colour := whitecolor ;
            background_colour := blackcolor ;
            grid_colour       := whitecolor ;
            cross_colour      := blackcolor ;
            x_y_values_col    := whitecolor ;
            graphdetails_col  := whitecolor ;
            axis_colour       := whitecolor ;
            numbers_colour    := whitecolor ;
            back^.colour       := whitecolor ;
            front^.colour      := whitecolor ;
            combo^.colour      := whitecolor ;
            front^.penpat      := black  ;
            back^.penpat       := dkgray ;
            combo^.penpat      := gray   ;
            front^.line_thickness := 1 ;
            back^.line_thickness  := 2 ;
            combo^.line_thickness := 2 ;
            sepia := false ;
          end ;{ make white against black monochrome  monitor.}
    end;{case}
  end ; { colourize }


  procedure Resize_to_screen ;
  begin
    setcursor(curslist[watchcursor]^^);
    if(front^.EXoffset <> 0)or(front^.WYoffset <> 0)or(front^.EXoffscale <>
  1)or
      (front^.WYoffscale <> 1) then
      begin
        this_front_saved := false ;
        normal_comb := false ;
        nlog_comb := false ;
        this_combo_saved := false ;
      end ;
    plot(remove,0,front);
    front^.EXoffset := 0 ;
    front^.WYoffset := 0 ;
    front^.EXoffscale := 1 ;
    front^.WYoffscale := 1 ;
    grid_and_axis ;
    if back^.set_to_display then plot(put_up,liftback,back) ;
    if combo^.set_to_display then plot(put_up,round(liftback/2),combo) ;
    if front^.set_to_display then plot(put_up,0,front);
    top_of_screen_details;  {on screen info re which graphs are read in}
  end ;


  procedure move( ItemNum : integer ) ;
  begin { move }
    setcursor(curslist[watchcursor]^^);
    this_front_saved := false ;
    this_combo_saved := false ;
    normal_comb := false ;
  nlog_comb := false ;
  if combo^.set_to_display then
    begin
      plot(remove,Round(liftback/2),combo);{ erase }
      combo^.set_to_display := false;
    end;
  if not wiped then
    begin
      plot(remove,0,front); { erase }
      wiped := true ;
    end ;
  case ItemNum of
   {Existing front offsets }
   {in SCRN UNTS + actual size of data at present EX the change }
{U} 1 : front^.WYoffset := front^.WYoffset + (back^.WYrange *
  percent_change/100) ; {add for up }
{D} 2 : front^.WYoffset := front^.WYoffset - (back^.WYrange *
  percent_change/100) ; {- for down }
{L} 3 : front^.EXoffset := front^.EXoffset - (back^.EXrange *
  percent_change/100) ; {- for left }
{R} 4 : front^.EXoffset := front^.EXoffset + (back^.EXrange *
  percent_change/100) ; {+ for right}
  end ; { case }
  if not OSEventAvail(mDownMask + keyDownMask, theEvent) then
    begin
      grid_and_axis ;
      if back^.set_to_display then plot(put_up,liftback,back) ;
      plot(put_up,0,front) ;
      wiped := false ;
    end ;
end ; { move }


procedure change_size( ItemNum : integer ) ;
begin { change_size }
  setcursor(curslist[watchcursor]^^);
  this_front_saved := false ;
  this_combo_saved := false ;
  normal_comb := false ;
  nlog_comb := false ;
  if combo^.set_to_display then
    begin
      plot(remove,Round(liftback/2),combo);{ erase }
      combo^.set_to_display := false;
    end;
  if not wiped then
    begin
      plot(remove,0,front); { erase }
      wiped := true ;
    end ;
  case ItemNum of
{T} 1 : front^.WYoffscale := front^.WYoffscale + percent_change/100 ;
{S} 2 : front^.WYoffscale := front^.WYoffscale - percent_change/100 ;
{W} 3 : front^.EXoffscale := front^.EXoffscale + percent_change/100 ;
{N} 4 : front^.EXoffscale := front^.EXoffscale - percent_change/100 ;
  end ; { case }
  if not OSEventAvail(mDownMask + keyDownMask, theEvent) then
    begin
      grid_and_axis ;
      if back^.set_to_display then plot(put_up,liftback,back)  ;
      plot(put_up,0,front) ;
      wiped := false ;
    end ;
end ; { change_size }


procedure graph_details ;
begin{ graph_details }
  if not info_up then
    begin{put info up}
      ClearWindow(MainPtr);
      penpat(black);
      forecolor(graphdetails_col) ;
      pensize(1,1) ;
      textsize(10);
      moveto(10,20);  drawstring( 'BACK GRAPH DETAILS ' ) ;
      if back^.is_read then begin { Back Graph Details }
          moveto(150,20);  drawstring( back^.line_zero);
          moveto(10,45);  drawstring( back^.line_one );
          moveto(10,65);  drawstring( back^.line_two );
          moveto(10,85); drawstring( back^.line_three );end;{ Details}
  moveto(10,115); drawstring( 'FRONT GRAPH DETAILS ' ) ;
      if front^.is_read then begin{ Front Graph Details }
          moveto(150,115); drawstring(front^.line_zero);
          moveto(10,140); drawstring( front^.line_one );
          moveto(10,160); drawstring( front^.line_two );
          moveto(10,180); drawstring(front^.line_three); end;{Details}
  moveto(10,210); drawstring( 'COMBINATION GRAPH' ) ;
      if combo^.is_read then begin { Combined Graph Details }
          moveto(150,210); drawstring( combo^.line_zero);
          moveto(10,235); drawstring( combo^.line_one );
          moveto(10,255); drawstring( combo^.line_two );end ;{Details}
  info_up := true ;
    end  {put info up}
  else begin{ replace with normal }
         redo_screen ; info_up := false ;
     end ;{ replace with normal }
end ;{ graph_details }


function answer_yes_no(str1,str2,str3,str4: string80): boolean;
var
  theItem    : integer;
  Yes_NoPtr :  DialogPtr;
begin{ answer_yes_no }
  SetCursor(Arrow);
  ParamText(str1,str2,str3,str4);  { and set up as parameter text   }
  Yes_NoPtr := getNewDialog(Yes_No_dlgID,NIL,Pointer(-1));{get d box}
  ModalDialog(NIL,theItem);         { put dialog box up; get result  }
  if theitem = 1 then answer_yes_no := true
  else answer_yes_no := false;
  DisposDialog(Yes_NoPtr);          { get rid of dialog box          }
end; { answer_yes_no }


procedure get_transform( var details_EX_trans,
                             details_WY_trans: transform_type;
                             transname       : string80 ;
                         var changes_made    : boolean        );
 var
   itemtype,  itemhit,  itemNo,  itoff,
   position       : Integer  ;
   ok_button      : boolean  ;
   trPtr          : DialogPtr;
   theItem        : Handle   ;
   display_rect   : Rect     ;
  EN_str          : str255   ;
  En_dial         : real     ;
  numbers_good    : boolean  ;

  procedure validate_button( itemhit: integer) ;
  var   itoff      : integer ;
        itemtype   : integer;
        theItem    : Handle;
        thecontrol: ControlHandle;
        display_rect: rect;

      procedure btnst( itemNo : integer ; on : boolean ) ;
      begin { btnst - set the state of button matching the ItemNo.}
        GetDItem(trPtr,itemNo,itemtype,theitem,display_rect);
        thecontrol:=controlhandle(theitem);
        if on then setctlvalue(thecontrol,1)
     else setctlvalue(thecontrol,0);
      end ; { btnst }

  begin{validate_button}
    case itemhit of
     11..19:begin for itoff:=11to 19do btnst(itoff,false);
      btnst(itemhit,true);end;
     21..29:begin for itoff:=21to 29do btnst(itoff,false);
      btnst(itemhit,true);end;
     31..39:begin for itoff:=31to 39do btnst(itoff,false);
      btnst(itemhit,true);end;
     41..49:begin for itoff:=41to 49do btnst(itoff,false);
      btnst(itemhit,true);end;
    end;{of case 1}
    case itemhit of
     11,16..19:begin
GetDItem(trPtr,20,itemtype,theitem,display_rect);
        SetIText(theitem,'');
          end;
     21,26..29:begin
        GetDItem(trPtr,30,itemtype,theitem,display_rect);
        SetIText(theitem,'');
          end;
     31,36..39:begin
        GetDItem( trPtr,40,itemtype,theitem,display_rect);
        SetIText(theitem,'');
          end;
     41,46..49:begin
        GetDItem( trPtr,50,itemtype,theitem,display_rect);
        SetIText(theitem,'');
          end;
    end;{of case 2}
    case itemhit of
      12..15: SelIText( trPtr, 20, 0, 6 );
      22..25: SelIText( trPtr, 30, 0, 6 );
      32..35: SelIText( trPtr, 40, 0, 6 );
      42..45: SelIText( trPtr, 50, 0, 6 );
    end;{of case 3}
  end;{validate_button}

  procedure fill_in_dial_box( this_trans: transform_type) ;
  var
    row, itemtype, the_button: integer ;
    theItem       : Handle  ;
    Str_for_dial : str255   ;
    display_rect : rect     ;
  begin {fill_in_dial_box}
```

```
      for row := 1 to 4 do
        begin{each row}
          case this_trans[row].ttype of
            tnone : the_button := row*10 +1 ;
            tadd  : the_button := row*10 +2 ;
            tminus: the_button := row*10 +3 ;
            ttimes: the_button := row*10 +4 ;
            tdiv  : the_button := row*10 +5 ;
            tlog  : the_button := row*10 +6 ;
            tln   : the_button := row*10 +7 ;
            talog : the_button := row*10 +8 ;
            taln  : the_button := row*10 +9 ;
          end;{of case}
          validate_button( the_button );
          case the_button of 12..15, 22..25, 32..35, 42..45:begin
              RealToStr(this_trans[row].n, Str_for_dial);
              GetDItem(trPtr,row*10+10,itemtype,theitem,display_rect);
              SetIText(theitem,Str_for_dial);    end; end;{of case 2}
        end;{each row}
      SelIText( trPtr, 20, 0, 6 );
    end ; {fill_in_dial_box}

    procedure get_values_to_save( var this_trans: transform_type;
                                  var  good_text: boolean        ) ;
    var
      item_look, row, itemtype, the_button : integer ;
      theItem       : Handle  ;
      dial_str      : str255  ;

        procedure assign_ttype( row, a_turned_on_item : integer  ) ;
        begin{assign_ttype}
          case a_turned_on_item of
            1:this_trans[row].ttype := tnone ;
            2:this_trans[row].ttype := tadd ;
            3:this_trans[row].ttype := tminus;
            4:this_trans[row].ttype := ttimes;
            5:this_trans[row].ttype := tdiv ;
            6:this_trans[row].ttype := tlog ;
            7:this_trans[row].ttype := tln ;
            8:this_trans[row].ttype := talog ;
            9:this_trans[row].ttype := taln ;
          end;{of case}
        end;{assign_ttype}

    begin{ get_values_to_save }
      row := 0;
      good_text := true;
      repeat{count through the rows as long as text returned is OK}
        begin{process the dialogue box row by row}
          row := row + 1;
          for item_look := 1 to 9 do
            begin{left to right of each row}
  GetDItem(trPtr,row*10+item_look,itemtype,
               theitem,display_rect);
              if GetCtlvalue( controlhandle(theitem) ) = 1 then
                begin{turned on button found}
                  assign_ttype(row, item_look);
                  the_button := row*10+item_look;
                end ; {turned on button found}
            end; {left to right of each row}
          case the_button of 12..15, 22..25, 32..35, 42..45:
            begin{text is available}
              GetDItem(trPtr,row*10+10,itemtype,theitem,display_rect) ;
            GetIText(theitem,dial_str) ;
            position := 1 ;
            this_trans[row].n:=rdreal(dial_str,position,ok);{str rel}
            if not ok then
              begin{text was no good}
                SysBeep(1);
                good_text := false ;
              end; {text was no good}
          end;{text is available}
        end;{of case for operator without number}
        case the_button of 11,16..19,21,26..29,31,36..39,41,46..49:
          begin{nil or log button pressed}
            GetDItem(trPtr,row*10+10,itemtype,theitem,display_rect) ;
            GetIText(theitem,dial_str) ;
            if dial_str <> '' then
              begin{ a number was entered, dut no button clicked }
                SysBeep(1);
                good_text := false ;
              end; { a number was entered, dut no button clicked }
          end;{nil or log button pressed}
        end;{of case for number without operator}
      end;{process the dialogue box row by row}
    until( row = 4 )or(not good_text);
  end ;{ get_values_to_save }


begin { get_transform }
  numbers_good := false ;
  paramtext(transname,'Modify X values.','','') ;
  trPtr := getNewDialog(trnsfrm_dlgID,NIL,Pointer(-1)) ;
  repeat{ may need to repeat if error checking finds a problem }
    begin
      fill_in_dial_box( details_EX_trans );
      repeat { interact with dialogue box }
         ModalDialog(NIL,itemhit);{ put dialog box up; get result}
         validate_button( itemhit );
      until ( itemhit = 1 ) or ( itemhit = 2 );{OK or cancel pushed}
      if itemhit = 1 then{if x's OK hit,save x detail, do y detail}
         get_values_to_save( details_EX_trans, numbers_good)
     else numbers_good := true ; { exit, preparing to cancel }
    end;
  until numbers_good;
  DisposDialog(trPtr);                     {get rid of x dialog box}
  numbers_good := false ;
  if itemhit = 1 then { if x's were OK ..do the y. }
    begin{ x's were ok, do the y dialogue }
      trPtr := getNewDialog(trnsfrm_dlgID,NIL,Pointer(-1)) ;
      repeat{ may need to repeat if error checking finds a problem }
        paramtext(transname,'Modify Y values.','','') ;
        fill_in_dial_box( details_WY_trans );
        repeat { interact with dialogue box }
            ModalDialog(NIL,itemhit); { put dialog box up get result}
            validate_button( itemhit );
        until ( itemhit = 1 ) or ( itemhit = 2 ) ;
        if itemhit = 1 then { y values choosen, ok pressed }
          get_values_to_save( details_WY_trans,numbers_good){y's OK?}
        else numbers_good := true ;      { exit, preparing to cancel }
      until numbers_good;
      DisposDialog(trPtr); { get rid of dialog box }
      changes_made := true ;
    end  { x's were ok, do the y dialogue }
  else
      if itemhit = 2 then changes_made := false ;
```

```
end ;{ get_transform }


procedure combine_nlog ;
var
  itemtype, position         : Integer;
  ok_button                  : boolean ;
  nLogDialPtr                : DialogPtr;
  theItem                    : Handle ;
  display_rectangle          : Rect ;
  EN_str                     : str255 ;
  En_dial                    : real ;
begin
  nLogDialPtr := getNewDialog(nLogdlgID,NIL,Pointer(-1)) ;
  ModalDialog(NIL,itemtype);      { put dialog box up, get result }
  if itemtype = 1 then
    begin{ ok button pressed}
      GetDItem(nLogDialPtr,3,itemtype,theitem,display_rectangle) ;
      GetIText(theitem,EN_str) ;
      position := 1 ;
      EN_dial := rdreal( EN_str, position, ok );{ Convert 1 str   }
      DisposDialog(nLogDialPtr);    { get rid of dialog box       }
      combine_charts( EN_dial ) ;
    end { ok button pressed}
  else
    begin { cancel button pressed }
      DisposDialog(nLogDialPtr);    { get rid of dialog box       }
      redo_screen ;                 { return to previous scree    }
    end ;{ cancel button pressed }
end ;{ combine_nlog }

procedure save_question( var confirmed: boolean );
 var
  itemhit, itemtype, position : Integer;
  ok_button, cancel, fr_button, cm_button : boolean ;
  saveDialPtr                 : DialogPtr;
  theItem                     : Handle ;
  display_rect                : Rect ;
  EN_str                      : str255 ;
  En_dial                     : real ;
  thecontrol                  : ControlHandle;

  procedure btnst( itemNo : integer ; on : boolean ) ;
  begin { btnst - set the state of the button matching the ItemNo. }
    GetDItem(saveDialPtr,itemNo,itemtype,theitem,display_rect);
    thecontrol:=controlhandle(theitem);
    if on then setctlvalue(thecontrol,1)
    else setctlvalue(thecontrol,0);
  end ; { btnst }

begin
  saveDialPtr := getNewDialog(SaveDialoguID,NIL,Pointer(-1)) ;
  ok_button := false;
  cancel := false;
  if not this_front_saved then
      fr_button := false
  else
    begin
      btnst( 2, true );
      fr_button := true;
    end;
  if( (normal_comb OR nlog_comb)and(not this_combo_saved) )then
      cm_button := false
  else
    begin
      btnst( 3, true );
      cm_button := true;
    end;
  repeat
      ModalDialog(NIL,itemhit);     { put dialog box up; get result}
      if itemhit = 2 then
        begin{ front save hit }
          if fr_button then SysBeep(1)
          else begin
              save_foreground;
              if this_front_saved then begin
                  btnst( itemhit, true ); fr_button := true; end;
              SetCursor(Arrow);
              if cm_button and fr_button then ok_button := true; end;
        end  { front save hit }
      else if itemhit = 3 then
        begin{ combo save hit }
          if cm_button then SysBeep(1)
          else begin
              save_combination;
              if this_combo_saved then begin
                  btnst( itemhit, true ); cm_button := true; end;
              SetCursor(Arrow);
              if fr_button and cm_button then ok_button := true; end;
        end  { combo save hit }
      else if itemhit = 1 then ok_button := true { ok to quit hit }
      else if itemhit = 4 then cancel := true; { cancel Quit hit }
  until ok_button or cancel;       { ok button pressed      }
  DisposDialog(saveDialPtr);       { get rid of dialog box  }
  confirmed := not cancel;
end;{ save question }

procedure equivalent_axis_values( event_point: point );
var
  mouse_point: point; equiv_axis_x, equiv_axis_y: real;
  in_from_left, up_from_bottom: integer;
begin{equivalent_axis_values}
  if (back^.is_read) and (overlaid) then
    begin{ok to proceed with process}
      MainPeek := WindowPeek(MainPtr);{get pointer to window record}
      mouse_point.h:=event_point.h-MainPeek^.contRgn^^.rgnBBox.left;
      mouse_point.v:=event_point.v-MainPeek^.contRgn^^.rgnBBox.top ;
      pensize(3,3);forecolor(background_colour);
      penpat(black);textsize(11);
      moveto( cross_point.h-6, cross_point.v-1 ); { erase }
      lineto( cross_point.h+4, cross_point.v-1 ); { previous }
      moveto( cross_point.h-1, cross_point.v+4 ); { cross }
      lineto( cross_point.h-1, cross_point.v-6 );
      pensize(40,1);     { wipe a rectangle for the x,y values}
      moveto(EXwindowdots-38,20);lineto(EXwindowdots-38,100);
      cross_point.h := mouse_point.h;
      cross_point.v := mouse_point.v;
      pensize(3,3);forecolor(grid_colour);
      moveto( mouse_point.h-6, mouse_point.v-1 );
      lineto( mouse_point.h+4, mouse_point.v-1 );
      moveto( mouse_point.h-1, mouse_point.v+4 );
      lineto( mouse_point.h-1, mouse_point.v-6 );
      pensize(1,1);forecolor(cross_colour);
      moveto( mouse_point.h-5, mouse_point.v );
      lineto( mouse_point.h+5, mouse_point.v );
      moveto( mouse_point.h, mouse_point.v+5 );
```

```
        lineto( mouse_point.h, mouse_point.v-5 );
        forecolor(x_y_values_col);
        moveto( EXwindowdots-37, 30 );
        drawstring('Mouse');
        moveto( EXwindowdots-37, 40 );
        drawstring('Click');
  { Take data,  Subtract back data minimum value, Scale it (if front),}
  { add offset (if front), ROUND data , add x data or subtract y data,}
  { Window scale the lot, and add window pixel position offset to it. }
  { Note prt_x_mv_pixel & prt_y_mv_pixel are 0 except when printing.  }
  { xplot := in_from_left                                             }
  {  + round( (chart^.EXoffset + (chart^.EX[i] - back^.EXmin)         }
  {        * chart^.EXoffscale ) * window_EX_scale ) ;                }
  {yplot := up_from_bottom                                            }
  {  - round( (chart^.WYoffset + (chart^.WY[i] - back^.WYmin)         }
  {        * chart^.WYoffscale) * window_WY_scale ) ;                 }
  { Therefore to get axis values from a mouse click do the following: }
  { Take mouse point h or v, Subtract window pixel position offset,   }
  { if x leave positive, but make y negative,                         }
  { UN-window-scale mouse point, then add back data min value.        }
        moveto( EXwindowdots-37, 60 );
        drawstring('X scale');
{7.5%in}in_from_left:=Round(EXwindowdots*((1.0-useEXscreen)/2 ));
equiv_axis_x := (mouse_point.h - in_from_left )/window_EX_scale
        + back^.EXmin;
        number_stamp( equiv_axis_x, 0, EXwindowdots-37, 70 );
        moveto( EXwindowdots-37, 90 );
        drawstring('Y scale');
        up_from_bottom:=Round(WYwindowdots*(useWYscreen+(1.0-
        useWYscreen)/2));
        equiv_axis_y := ( up_from_bottom - mouse_point.v )
               /window_WY_scale + back^.WYmin;
        number_stamp( equiv_axis_y, 0, EXwindowdots-37, 100 );
      end;{ok to proceed with process}
  end;{equivalent_axis_values}


procedure invoke_the_list_manager ;
var
  rView, databounds                     : Rect;
  cSize, theCell, MLoc                  : Point;
  theProc, WLoc, i, modifiers           : INTEGER;
  value_as_real                         : REAL;
  value_as_string                       : string;
  theWindow                             : WindowPtr;
  drawIt,hasGrow,scrollHoriz,scrollVert,
  clicked_in_list, double_clicked       : BOOLEAN;
  dataviewHDL                           : ListHandle;
begin { invoke_the_list_manager }
  list_up := true ; { set global boolean for menu grey control }
  double_clicked := false;
  (*theWindow := MainPtr;*)
  theWindow := DatWPtr;
  SetPort(DatWPtr); {DatWPtr^.txFont := 2;}
  SelectWindow(DatWPtr);                { and make window active }
  SetCursor(CursList[watchCursor]^^);   { set cursor to watch    }
  textsize(12); forecolor(background_colour);
  pensize(1,29);moveto(1,1);lineto(220,1);pensize(2,2);
  forecolor(foreground_colour);
  moveto(20,10); drawstring(' Back Chart       Front Chart');
  moveto(20,25);drawstring('   X          Y            X          Y');
  moveto(1,28);lineto(219,28);lineto(219,1);
  rView.top := 30; rView.left := 10;
rView.bottom := 290; rView.right := 205 ;
databounds.top := 1 ; databounds.left := 1;
databounds.bottom := 258; databounds.right := 5 ;
cSize.h := 50; cSize.v := 15;{Size of cell not specified, so deflt}
theProc := 0;
drawIt := true;
hasGrow := true;
scrollHoriz := false; scrollVert := true;
dataviewHDL := LNew (rView, databounds, cSize, theProc, theWindow,
       drawIt, hasGrow, scrollHoriz, scrollVert);
for theCell.v := 1 to 256 do
  begin{ jump to next row, fill in an element in each column }
    theCell.h := 1;
    LClrCell(theCell,dataviewHDL);
    i := theCell.v;
    value_as_real := back^.EX[i];
    RealToStr( value_as_real, value_as_string );
    LSetCell(@value_as_string[1], 5, theCell, dataviewHDL);
    theCell.h := 2;
    LClrCell(theCell,dataviewHDL);
    LClrCell(theCell,dataviewHDL);
    value_as_real := back^.WY[i];
    RealToStr( value_as_real, value_as_string );
    LSetCell(@value_as_string[1], 5, theCell, dataviewHDL);
    theCell.h := 3;
    LClrCell(theCell,dataviewHDL);
    i := theCell.v;
    if front^.is_read then
      begin{ put in front listing }
        value_as_real := front^.EX[i];
        RealToStr( value_as_real, value_as_string );
        LSetCell(@value_as_string[1], 5, theCell, dataviewHDL);
      end; { put in front listing }
    theCell.h := 4;
    LClrCell(theCell,dataviewHDL);
    if front^.is_read then
      begin{ put in front listing }
        value_as_real := front^.WY[i];
        RealToStr( value_as_real, value_as_string );
        LSetCell(@value_as_string[1], 5, theCell, dataviewHDL);
      end; { put in front listing }
  end; { jump to next row }
FlushEvents(everyEvent,0);    { clear events from previous state  }
SetCursor(Arrow);
repeat
  if OSEventAvail(mDownMask + keyDownMask, theEvent) then
    begin{ an Operating System event! }
      MLoc  := theEvent.Where;          { get mouse position      }
      WLoc := FindWindow(MLoc,theWindow);{get window,loc in window }
      MLoc.v := MLoc.v - 35 ;
      if(theWindow = DatWPtr)and(theEvent.What = mouseDown)then
        begin
          double_clicked := LClick(MLoc,modifiers,dataviewHDL);
          FlushEvents(everyEvent,0);
        end;
    end; { an Operating System event! }
until double_clicked;
LDispose( dataviewHDL );
list_up := false;
theWindow := MainPtr;
SetPort(MainPtr);
SelectWindow(MainPtr);          { and make window active }
FlushEvents(everyEvent,0);{clear events - prevents cross appearing}
```

```
  redo_screen;
  FlushEvents(everyEvent,0);{clear events - prevents cross appearing}
end;  { invoke_the_list_manager }


{$S macstuff}

{    *********   items in Apple Menu   ***********  }

procedure DoAbout;
{ purpose         bring up 'About...' box using a dialog box }
var
  theItem         : Integer;
  AboutPtr        : DialogPtr;
  S1,S2,S3        : StringHandle;
begin
  SetCursor(CursList[myCursor]^^);  { set to my cursor                 }
  ShowCursor;                       { and turn it back on              }
  S1 := GetString(Text1ID);         { get text from resource file      }
  S2 := GetString(Text2ID);
  S3 := GetString(Text3ID);
  ParamText(S1^^,S2^^,S3^^,'');     { and set up as parameter text     }
  AboutPtr := getNewDialog(AboutID,NIL,Pointer(-1));{get dialog box   }
  ModalDialog(NIL,theItem);         { put dialog box up; get result    }
  DisposDialog(AboutPtr);           { get rid of dialog box            }
  Redo_screen;
  SetCursor(Arrow);
end; { of proc DoAbout }

procedure DoDeskAcc(Item : Integer);
{ purpose         start up desk accessory from Apple menu }
var
  SavePort            : GrafPtr;
  RefNum              : Integer;
  DName               : String;
begin
  GetPort(SavePort);                { save port before starting it  }
  GetItem(MenuList[AppleM],Item,DName);{ get name of desk accessory }
  refNum := OpenDeskAcc(DName);     { and start that sucker up!     }
  SetPort(SavePort);                { restore grafport and continue }
end; { of proc DoDeskAcc }

{    *********   event handling routines   ***********  }

procedure ToggleFlag(var Flag : Boolean; Mndx,Indx : Integer);
{ purpose   checks or de-checks item Indx in menu Mndx last update 20 Aug 86}
var
  Ch              : Char;
begin
  Flag := not Flag;                 { toggle flag (for you)          }
  if Flag                           { if flag is True...             }
    then Ch := Chr(CheckMark)       {   then check item in menu      }
    else Ch := Chr(NoMark);         {   else clear any checkmark     }
  SetItemMark(MenuList[Mndx],Indx,Ch){ put char by item in menu      }
end; { of proc ToggleFlag }

procedure SetItemState(Mndx,Indx : Integer; Flag : Boolean);
{ purpose   if true, enables item Indx of menu Mndx; else disables  last
update  22 Aug 86 }
begin
  if Flag
    then EnableItem (MenuList[Mndx],Indx)
    else DisableItem(MenuList[Mndx],Indx)  ;
  checkitem(menulist[Mndx],Indx,false);  {uncheck every menu item}
end; { of proc SetItemState }

procedure SetItemcheck(Mndx,Indx : Integer);
{ purpose  sets the check mark next to item }
begin
  Checkitem(menulist[Mndx],indx,true);
end; { of proc SetItemcheck }


procedure UpdateMenu;{ enable or disable items in I/O menu as needed}
var   i : integer ;
begin { UpdateMenu }
  SetItemState(FileM,1,not list_up );              { read foregrd }
  if back^.is_read then setitemcheck(FileM,1);
  SetItemState(FileM,2,back^.is_read and not list_up);{ read backgrd }
  if front^.is_read then setitemcheck(FileM,2);
  SetItemState(FileM,3,not info_up and not list_up);
  if not( (back_EX_trans[1].ttype = tnone)and
     (back_EX_trans[2].ttype= tnone)and
     (back_EX_trans[3].ttype = tnone)and
     (back_EX_trans[4].ttype = tnone)and
     (back_WY_trans[1].ttype = tnone)and
     (back_WY_trans[2].ttype = tnone)and
     (back_WY_trans[3].ttype = tnone)and
     (back_WY_trans[4].ttype = tnone) )
      then
    setitemcheck(FileM,3);
  SetItemState(FileM,4,not info_up and not list_up);
  if not( (front_EX_trans[1].ttype = tnone)and
     (front_EX_trans[2].ttype= tnone)and
     (front_EX_trans[3].ttype = tnone)and
     (front_EX_trans[4].ttype = tnone)and
     (front_WY_trans[1].ttype = tnone)and
     (front_WY_trans[2].ttype = tnone)and
     (front_WY_trans[3].ttype = tnone)and
     (front_WY_trans[4].ttype = tnone) )
      then
        setitemcheck(FileM,4);
  SetItemState(FileM,5,false);{ Item 5,line only,Item 6 save foregrd}
  SetItemState(FileM,6,back^.is_read and front^.is_read and
                       not list_up);
  if this_front_saved then setitemcheck(FileM,6);
  SetItemState(FileM,7,combo^.is_read and not list_up);{save comb ch}
  if this_combo_saved then setitemcheck(FileM,7);
  SetItemState(FileM,8,not info_up and not list_up);
  if not( (save_EX_trans[1].ttype = tnone)and
     (save_EX_trans[2].ttype= tnone)and
     (save_EX_trans[3].ttype = tnone)and
     (save_EX_trans[4].ttype = tnone)and
     (save_WY_trans[1].ttype = tnone)and
     (save_WY_trans[2].ttype = tnone)and
     (save_WY_trans[3].ttype = tnone)and
     (save_WY_trans[4].ttype = tnone) )
  then
        setitemcheck(FileM,8);
  SetItemState(FileM,9,not info_up and not list_up);
  if not( (combo_EX_trans[1].ttype = tnone)and
     (combo_EX_trans[2].ttype= tnone)and
     (combo_EX_trans[3].ttype = tnone)and
     (combo_EX_trans[4].ttype = tnone)and
     (combo_WY_trans[1].ttype = tnone)and
     (combo_WY_trans[2].ttype = tnone)and
```

```
   (combo_WY_trans[3].ttype = tnone)and
   (combo_WY_trans[4].ttype = tnone) )
then
      setitemcheck(FileM,9);
 SetItemState(FileM,10,false);                           { line only }
 SetItemState(FileM,11,back^.is_read and not info_up and
                 not list_up); { print}
 SetItemState(FileM,12,false);                           { line only }
 SetItemState(FileM,13,true);                      { Can always QUIT }

 SetItemState(EditM,0,back^.is_read and not info_up );
 SetItemState(EditM,6,back^.is_read and not info_up );
 if list_up then setitemcheck(EditM,6);

 SetItemState(ScreenM, 0, back^.is_read  and not list_up);
 SetItemState(ScreenM, 1, (back^.is_read or front^.is_read) and
          not list_up); { Put up graph details }
 if info_up then setitemcheck(screenM,1);
 SetItemState(ScreenM, 2, false);
 SetItemState(ScreenM, 3, back^.is_read and front^.is_read and not
 info_up and not list_up);{overlaid / split  }
 SetItemState(ScreenM, 4, back^.is_read and not info_up and not
 list_up and not(overlaid and comb_just_put_up) );
 if back^.showing then setitemcheck(screenM,4);
 SetItemState(ScreenM, 5, back^.is_read and front^.is_read and not   info_up
and not list_up and not(overlaid and comb_just_put_up));
 if front^.showing then setitemcheck(screenM,5);
 SetItemState(ScreenM, 6, combo^.is_read and not info_up and not
 list_up and not(overlaid and comb_just_put_up) );
 if combo^.showing then setitemcheck(ScreenM,6);
 SetItemState(ScreenM, 7, false);
 SetItemState(ScreenM,8,back^.is_read and not info_up and
                not list_up);{grid}
 if grid_up then setitemcheck(screenM,8);
 SetItemState(ScreenM, 9, back^.is_read and not info_up and not
     list_up and overlaid);{axis}
 if axis_up then setitemcheck(screenM,9);

 SetItemState(MoveM,0, front^.is_read and front^.set_to_display and
        not info_up and not list_up);
 SetItemState(MoveM,1, front^.is_read and front^.set_to_display and
 overlaid and not info_up and not list_up );
 SetItemState(MoveM,2, front^.is_read and front^.set_to_display and
 overlaid and not info_up and not list_up);
 SetItemState(MoveM,3, front^.is_read and front^.set_to_display and
     not info_up and not list_up);
 SetItemState(MoveM,4, front^.is_read and front^.set_to_display and
   not info_up and not list_up);
 SetItemState(MoveM,5, false);
 SetItemState(MoveM,6, front^.is_read and front^.set_to_display and
   not info_up and not list_up);
 SetItemState(MoveM,7, front^.is_read and front^.set_to_display and
   not info_up and not list_up);
 SetItemState(MoveM,8, front^.is_read and front^.set_to_display and
   not info_up and not list_up);
 SetItemState(MoveM,9, front^.is_read and front^.set_to_display and
   not info_up and not list_up);
 SetItemState(MoveM,10,false);
 SetItemState(MoveM,11,front^.is_read and front^.set_to_display and
   not info_up and not list_up);{wrap left}
 SetItemState(MoveM,12,front^.is_read and front^.set_to_display and
                       not info_up and not list_up);{wrap right}
```

```
  SetItemState(ResizeM,0, front^.is_read and front^.set_to_display and
       not info_up and not list_up);
  SetItemState(ResizeM,1, front^.is_read and front^.set_to_display and
     overlaid and not info_up and not list_up);
  SetItemState(ResizeM,2, front^.is_read and front^.set_to_display and
     overlaid and not info_up and not list_up);
  SetItemState(ResizeM,3, front^.is_read and front^.set_to_display and
         not info_up and not list_up);
  SetItemState(ResizeM,4, front^.is_read and front^.set_to_display and
         not info_up and not list_up);
  SetItemState(ResizeM,5, false);
  SetItemState(ResizeM,6, front^.is_read and front^.set_to_display and
         not info_up and not list_up );
  SetItemState(ResizeM,7, front^.is_read and front^.set_to_display and
         not info_up and not list_up);
  SetItemState(ResizeM,8, front^.is_read and front^.set_to_display and
         not info_up and not list_up );
  SetItemState(ResizeM,9, front^.is_read and front^.set_to_display and
         not info_up and not list_up);
  case percent_change of
   10 : begin setitemcheck(MoveM,6); setitemcheck(ResizeM,6); end ;
    5 : begin setitemcheck(MoveM,7); setitemcheck(ResizeM,7); end ;
    2 : begin setitemcheck(MoveM,8); setitemcheck(ResizeM,8); end ;
    1 : begin setitemcheck(MoveM,9); setitemcheck(ResizeM,9); end ;
  end ;{case to set tick on percent}
  SetItemState(ResizeM,10, false);
  SetItemState(ResizeM,11, back^.is_read and front^.is_read and
                not info_up and not list_up );
  SetItemState(ResizeM,12, false);
  SetItemState(ResizeM,13, front^.is_read and not info_up and
                not list_up );
  SetItemState(ResizeM,14, front^.is_read and not info_up and
                not list_up );
  SetItemState(AnalysisM,0, back^.is_read and front^.is_read and
                not info_up and not list_up);
  SetItemState(AnalysisM,1, back^.is_read and front^.is_read and
                not info_up and not list_up);
  if normal_comb then setitemcheck(AnalysisM,1) ;
  SetItemState(AnalysisM,2, back^.is_read and front^.is_read and
                not info_up and not list_up);
  if nlog_comb then setitemcheck(AnalysisM,2) ;
  SetItemState(AnalysisM,3, false );
  SetItemState(AnalysisM,4, back^.is_read and front^.is_read and
                not info_up and not list_up);
  if not((back_weight =1)and(front_weight = 1))then
   setitemcheck(AnalysisM,4);

  SetItemState(ColourM,0, not info_up and not list_up );
  SetItemState(ColourM,1, not info_up and not list_up );
  SetItemState(ColourM,2, not info_up and not list_up );
  SetItemState(ColourM,3, not info_up and not list_up );
  SetItemState(ColourM,4, not info_up and not list_up );
  SetItemState(ColourM,5, not info_up and not list_up );
  setitemcheck(ColourM,the_colour_choice);
end; { of proc UpdateMenu }


    procedure handle_file_menu( item : integer ) ;
    var
      read_was_ok,
      changes_made,
      confirmed      : boolean;
    begin{ handle_file_menu }
```

```
case item of
  1:begin{ reading a back chart }
      info_up := false ;    { in case graph details were up.}
      if back^.is_read then {If a back chart up, get rid of it}
          plot(remove,liftback,back) ;
      Read_chart_file( true, back, read_was_ok );{read backch}
      if (read_was_ok) and (front^.is_read) then
        begin
          ClearWindow(MainPtr);{ Empty window, resize screen }
          Resize_to_screen;{New back,-frt offset-0 &offscale 1}
        end
      else
          redo_screen ;
    end ;{ reading a back chart }
  2:begin{ reading a front chart }
      info_up := false ;   { in case graph details were up. }
      if front^.is_read then {If a front ch up, get rid of it}
          plot(remove,0,front) ;
      Read_chart_file( false, front, read_was_ok) ;
      redo_screen ;
    end ;{ reading a front chart }
  3:begin{choose back transformed}
      get_transform( back_EX_trans,back_WY_trans,
       'Alter Back Read.',changes_made);
      if changes_made then
        begin{ Valid changes made in dialogue box. }
          if answer_yes_no('   Do you want to set the',
                           '   FRONT Transformation',
                           '   the same as the',
                           '   BACK Tranformation ?' ) then
            begin
              front_EX_trans := back_EX_trans;
              front_WY_trans := back_WY_trans;
            end
        end;{ Valid changes made in dialogue box. }
      redo_screen ;
    end;{choose back transformed}
  4:begin{choose front transformed}
          if answer_yes_no('   Do you want to set the',
                           '   FRONT Transformation',
                           '   the same as the',
                           '   BACK Tranformation ?' ) then
            begin
              front_EX_trans := back_EX_trans;
              front_WY_trans := back_WY_trans;
              redo_screen ;
            end
          else
            begin
              get_transform( front_EX_trans,front_WY_trans,
        'Alter Front Read.',changes_made);
              redo_screen; { repair previous screen }
            end;
    end;{choose front transformed}
  5: ; { dummy }
  6:begin{save front chart}
      info_up := false ;    { in case graph details were up.}
      Save_Foreground;
      redo_screen ;
    end ;{save front chart}
  7:begin{save combo chart}
      info_up := false ;    { in case graph details were up.}
      Save_Combination;
```

```
          redo_screen ;
        end ;{save combo chart}
      8: begin
           get_transform( save_EX_trans,save_WY_trans,
           'Alter Front Save.',changes_made);
           redo_screen; { repair previous screen }
         end;
      9: begin
           get_transform( combo_EX_trans,combo_WY_trans,
         'Alter Combination Save.',changes_made);
           redo_screen; { repair previous screen }
         end;
     10: ; { dummy }
     11: begin do_print; redo_screen; end ;
     12: ;
     13: begin {QUIT}(*if any_changes_to_file, Save -   booleans*)
           if((normal_comb OR nlog_comb)and(not this_combo_saved))
             or(not this_front_saved) then
               if front^.is_read then save_question(confirmed);
           if confirmed then finished := true
           else redo_screen;
         end;  {QUIT}
    end ; { case }
  end ;{ handle_file_menu }

  procedure handle_edit_menu( item : integer ) ;
  begin { handle_edit_menu }
    case item of
      1,2,3,4,5: ;
      6 : invoke_the_list_manager ;
    end;{of case}
  end;  { handle_edit_menu }

  procedure handle_Screen_menu( item : integer ) ;
    begin { handle_Screen_menu }
      case Item of
         1 : graph_details ;
         2 : ; { dummy }
         3 : if overlaid and comb_just_put_up then
                 redo_screen
             else
                 if overlaid then overlay(false)
     else overlay(true);{ overlay or split b,f,c }
         4 : if back^.is_read then
                begin{ turn back off or on }
                 if back^.set_to_display then
back^.set_to_display:=false
                 else back^.set_to_display := true ;
                 if not overlaid or back^.set_to_display then
                     plot(back^.set_to_display,liftback,back)
                 else redo_screen ;
                 top_of_screen_details;{on screen info for graphs}
                end; { turn back off or on }
         5 : if front^.is_read then
                begin{ turn front off or on }
                 if front^.set_to_display then
front^.set_to_display:=false
                 else front^.set_to_display := true ;
                 if not overlaid or front^.set_to_display then
                     plot(front^.set_to_display,0,front)
                 else redo_screen ;
                 top_of_screen_details;
                end; { turn front off or on }
```

```
            6 : if combo^.is_read then
                  begin{ turn combo off or on }
                    if combo^.set_to_display then
combo^.set_to_display:=false
                    else combo^.set_to_display := true;
                    if not overlaid or combo^.set_to_display then
                   plot(combo^.set_to_display,round(liftback/2),combo)
                    else redo_screen ;
                    top_of_screen_details;
                  end ;{ turn combo off or on }
            7 : ; { dummy }
            8 : begin { grid sequence }
                  if grid_up then grid_up := false
                  else            grid_up := true ;
                  redo_screen
                end ; { grid sequence }
            9 : begin { axis sequence }
                  if axis_up then axis_up := false
                  else axis_up := true ;
                  redo_screen ;
                end ;  { axis sequence }
         end; {case}
      end ;{ handle_Screen_menu }

      procedure handle_move_menu( item: integer ) ;
      begin { handle_move_menu }
         case Item of
            1,2,3,4 : move( Item ) ;
            6 : percent_change := 10 ; { reset change constant}
            7 : percent_change := 5 ;  { - also used for wrap }
            8 : percent_change := 2 ;  { reset change constant}
            9 : percent_change := 1 ;  { - also used for wrap }
           10 : ; { dummy }
           11 : wrap_left ;{ if split, dont clearscreen }
           12 : wrap_right ;{ if split, dont clearscreen }
         end;{ case }
      end ; { handle_move_menu }

      procedure handle_resize_menu( item : integer ) ;
      begin { handle_resize_menu }
         case Item of
            1,2,3,4 : change_size( Item ) ;
            6 : percent_change := 10 ; { reset change constant }
            7 : percent_change := 5 ;  {  also used for wrap   }
            8 : percent_change := 2 ;  { reset change constant }
            9 : percent_change := 1 ;  {  also used for wrap   }
           10 : ; { dummy }
           11 : Resize_to_screen ;{reset front to size it began at}
         end;{ case }
      end ; { handle_resize_menu }

     procedure handle_analysis_menu( item : integer ) ;
     var tick : boolean ;
     begin { handle_analysis_menu }
       tick := false ;
       case Item of
         1 : combine_charts( 0.0 ) ;
         2 : combine_nlog ; { combine nlogged data }
         3 : ; { dummy }
         4 : combining_ratio ;{ back_weight:front_we normally 1:1 }
       end;{case}
     end ; { handle_analysis_menu }
```

```
      procedure handle_scale_menu( item: integer );{change colours}
      begin
        colourize( item );{ change the displayed colours }
        redo_screen ;
      end;

  procedure HandleMenu(MenuInfo : LongInt);
  { purpose         decode MenuInfo and carry out command }
  var
    Menu          : Integer;     { menu number that was selected     }
    Item          : Integer;     { item in menu that was selected    }
    B             : Boolean;     { dummy flag for SystemEdit call    }
  begin
    SetPort(MainPtr);            { set window to current graf port   }
    SelectWindow(MainPtr);{ make window active -ie using calc, etc. }
    FrontWindow := MainPtr;      { remember that it's in front        }
    if MenuInfo <> 0 then begin
      PenNormal;                 { set the pen back to normal         }
      Menu := HiWord(MenuInfo); { find which menu the command is in }
      Item := LoWord(MenuInfo); { get the command number              }
      case Menu of               { and carry it out                    }
        ApplMenu     : if Item = 1
                         then DoAbout   { bring up "About..."window }
                         else DoDeskAcc(Item);{start desk accessory }
        FileMenu     : handle_file_menu(item) ;
        EditMenu     : handle_edit_menu(item) ;
        ScreenMenu   : handle_Screen_menu(item) ;
        MoveMenu     : handlè_move_menu(item) ;
        ResizeMenu   : handle_resize_menu(item) ;
        AnalysisMenu : handle_analysis_menu( item ) ;
        ColourMenu   :  handle_scale_menu( item ) ;
      end;{case of Menu}
      HiliteMenu(0);              { reset menu bar                      }
      UpdateMenu;                 { make any changes needed             }
    end ;
  end; {of proc HandleMenu}

procedure HandleClick(WPtr : WindowPtr; MLoc : Point);
{  purpose         handle mouse click within window }
begin
  if WPtr = MainPtr               { if this is our window...   }
    then if WPtr <> FrontWindow   { and it's not in front^... }
      then SelectWindow(WPtr)     { ...then make it active     }
end; { of proc HandleClick }

procedure HandleGoAway(WPtr : WindowPtr; MLoc : Point);
{ purpose         handle mouse click in go-away box }
var
  WPeek              : WindowPeek;    { for looking at windows       }
begin
  if WPtr = FrontWindow then begin  { if it's the active window   }
    WPeek := WindowPeek(WPtr);       { peek at the window           }
    if TrackGoAway(WPtr,MLoc) then begin {and the box is clicked }
      if WPeek^.WindowKind = userKind then{ if it's our window    }
        begin
          Finished := True ;              { then time to stop      }
            if any_changes_to_file then Save_Combination;
        end
      else CloseDeskAcc(WPeek^.WindowKind) { else close DeskAcc   }
    end
  end
  else SelectWindow(WPtr)                 { else make it active }
end; { of proc HandleGoAway }
```

```
procedure HandleGrow(WPtr : WindowPtr; MLoc : Point);
{   purpose          handle mouse click in grow box }
type
  GrowRec            =
    record
      case Integer of
        0            : (Result       : LongInt);
        1            : (Height,Width : Integer)
    end;
var
  GrowInfo           : GrowRec;
  test : real ;
begin
  if WPtr = MainPtr then with GrowInfo do
    begin { if it's our window     }
      Result := GrowWindow(WPtr,MLoc,GrowArea); { get amt of growth }
      SizeWindow(WPtr,Width,Height,True);       { resize window     }
      if (height<>0)and(width<>0)then {No move growicon,hght,wdth=0!}
       begin{ movement of grow icon recognized }
        EXwindowdots := Width ;  { Standard scales are for screen }
        WYwindowdots := Height ; { Use 90% of available window.   }
        window_EX_scale := useEXscreen *EXwindowdots/back^.EXrange;
         window_WY_scale := useWYscreen *WYwindowdots/back^.WYrange; {both
offset & range must be rescaled,both used in changes to offset}
front^.EXoffset:=front^.EXoffset*window_EX_scale/prev_EX_scale;
    front^.WYoffset:=front^.WYoffset*window_WY_scale/prev_WY_scale;
        front^.EXrange:=front^.EXrange*window_EX_scale/prev_EX_scale;
        front^.WYrange:=front^.WYrange*window_WY_scale/prev_WY_scale;
        prev_EX_scale := window_EX_scale ;
        prev_WY_scale := window_WY_scale ;
        InvalRect(WPtr^.portRect) ;           { set up for update }
        if not overlaid then {split, rescale as req.}
            begin  liftback := round( WYwindowdots / 2 );{split,redo}
                   window_WY_scale  := window_WY_scale / 2 ; end ;
        if info_up then { redo the graph details }
            begin  info_up := false ; { graph details will be redone }
                   graph_details; end { in case graph details up. }
        else redo_screen ; { redraw front, back and combo charts }
        penpat(black) ;
        pensize(0,0) ;
        DrawGrowIcon(WPtr);     { draw the grow box }
       end;{ movement of grow icon recognized }
    end ; {  if it's our window    }
end; { of proc HandleGrow }

procedure DoMouseDown(theEvent:EventRecord);
{  purpose         identify where mouse was clicked and handle it }
var
  Location      : Integer;
  theWindow     : WindowPtr;
  MLoc          : Point;
  WLoc          : Integer;
begin
  MLoc   := theEvent.Where;           { get mouse position            }
  WLoc := FindWindow(MLoc,theWindow);{ get window, loc in window      }
  case WLoc of                        { handle window locations       }
    InMenuBar   : HandleMenu(MenuSelect(MLoc));  { in the menu        }
    InContent   : HandleClick(theWindow,MLoc);   { inside the window  }
    InGoAway    : HandleGoAway(theWindow,MLoc);  { in the go away box}
    InGrow      : HandleGrow(theWindow,MLoc);    { in the grow box    }
    InDrag     : DragWindow(theWindow,MLoc,DragArea);{in the drag bar}
    InSysWindow : SystemClick(theEvent,theWindow) { in a DA window    }
  end
end; { of proc DoMouseDown }

procedure DoMouseUp(theEvent:EventRecord);
{  purpose         identify where mouse was clicked and handle it }
var
  Location          : Integer;
  theWindow         :  WindowPtr;
  MLoc              :  Point;
  WLoc              :  Integer;
begin
  MLoc  := theEvent.Where;                { get mouse position        }
  WLoc := FindWindow(MLoc,theWindow); { get window, loc in window  }
  case WLoc of                            { handle window locations   }
    InMenuBar   : ;
    InContent   : if not list_up then equivalent_axis_values(MLoc) ;
    InGoAway    : ;
    InGrow      : ;
    InDrag      : ;
    InSysWindow : ;
  end;{case}
end; { of proc DoMouseUp }

procedure DoKeypress(theEvent : EventRecord);
{ purpose         handles keypress (keyDown, autoKey) event }
var
  KeyCh             : Char;
begin
  if (theEvent.modifiers and cmdKey) <> 0 then begin{menu key command}
    KeyCh := Chr(theEvent.Message and charCodeMask);{decode character}
    HandleMenu(MenuKey(KeyCh))                       { get menu and item}
  end
  else SysBeep(1)                                    { do *something*    }
end; { of proc DoKeypress }

procedure DoUpdate(theEvent : EventRecord);
{ purpose         handles window update event }
var
  SavePort,theWindow    : WindowPtr;
begin
  theWindow := WindowPtr(theEvent.Message);  { find which window      }
  if theWindow = MainPtr then begin          { only update ours       }
    SetCursor(CursList[watchCursor]^^);      { set cursor to watch    }
    GetPort(SavePort);                       { save current grafport  }
    SetPort(theWindow);                      { set as current port    }
    BeginUpdate(theWindow);                  { signal start of update}
  { and here's the update stuff! }
  { ClearWindow(theWindow);     }            { do update stuff        }
  { now, back to our program...}
    EndUpdate(theWindow);                    { signal end of update   }
    SetPort(SavePort);                       { restore grafport       }
    SetCursor(Arrow)                         { restore cursor         }
  end
end; { of proc DoUpdate }

procedure DoActivate(theEvent : EventRecord);
var     { purpose  DoActivate handles window activation event. }
  eI : Integer;  AFlag : Boolean; theWindow : WindowPtr;
begin
  with theEvent do begin
    theWindow := WindowPtr(Message);         { get the window         }
    AFlag := Odd(Modifiers);                 { get activate/deactive }
    if AFlag then begin                      { if it's activated...   }
```

```
      SetPort(theWindow);                         {   make it the port    }
      FrontWindow := theWindow;                   {   know it's in front  }
      penpat(black) ;
      pensize(0,0) ;
      DrawGrowIcon(theWindow);                    {   set size box        }
    end
    else begin
      SetPort(ScreenPort);                        { else reassign port    }
      if theWindow = FrontWindow                  { if it's in front      }
        then FrontWindow := NIL                   { ...then forget that   }
    end;
    if theWindow = MainPtr then begin             { if it's our window    }
     (* SetItemState(EditM,1,not AFlag);          {   update edit cmds    }
      for eI := 3 to 6 do
        SetItemState(EditM,eI,not AFlag);
      SetItemState(EditM,8,AFlag);*)              {   update Quit command }
     (* for eI := IM to ResizeM do                {   update other menus  }
        SetItemState(eI,0,AFlag);*)
      DrawMenuBar                                 {   update menu bar     }
    end
  end
end; { of proc DoActivate }

procedure Initialize;{ purpose initialize everything for the program }
var
  Indx, i       : Integer;
  Result        : Real;
begin
  { Set up turbo's standard file type and file creator }
  Filetype    := 'BINA';
  Filecreator := 'Gmat';
  Texttype    := 'TEXT';
  TextCreator := 'Gmat';
  { initialize all the different managers               }
  InitGraf(@thePort);       { create a grafport for the screen  }
  InitFonts;                { start up the font manager         }
  InitWindows;              { start up the window manager       }
  InitMenus;                { start up the menu manager         }
  TEInit;                   { start up the text manager for DAs }
  InitDialogs(NIL);         { start up the dialog manager       }
  FlushEvents(everyEvent,0);{ clear events from previous state  }
  { Get  four  standard  system  cursors, plus  one  custom one.}
  for Indx := iBeamCursor to watchCursor do begin
    CursList[Indx]:=GetCursor(Indx);{ read from system resource }
    HLock(Handle(CursList[Indx]))    { lock the handle down     }
  end;
  CursList[myCursor] := GetCursor(MyCursID);{get cursor from res}
  HLock(Handle(CursList[myCursor]));          { and lock it down }
  SetCursor(CursList[watchCursor]^^);{ bring up watch cursor    }
  { set up menus }
  MenuList[AppleM]   := GetMenu(ApplMenu);{read menus from resfork}
  MenuList[FileM]    := Getmenu(FileMenu);
  MenuList[EditM]    := Getmenu(EditMenu);
  MenuList[ScreenM]  := GetMenu(ScreenMenu);
  MenuList[MoveM]    := GetMenu(MoveMenu);
  MenuList[ResizeM]  := GetMenu(ResizeMenu);
  MenuList[AnalysisM]  := GetMenu(AnalysisMenu);
  MenuList[ColourM]  := GetMenu(ColourMenu);
  AddResMenu(MenuList[AppleM],'DRVR');{pull in all desk accessories}
  for Indx := 1 to MenuCnt do          {  place menus in menu bar  }
    InsertMenu(MenuList[Indx],0);
  DrawMenuBar;                    { draw updated menu bar to screen  }
  { set up window stuff }
```

```
GetWMgrPort(ScreenPort);       { get grafport for all windows      }
SetPort(ScreenPort);           { and keep hand just in case        }
MainPtr := GetNewWindow(MainID,@MainRec,Pointer(-1));{get window }
DatWPtr := GetNewWindow(DatWID,@DatWRec,Pointer(-1));{get window }
SetPort(MainPtr);             { set window to current graf port   }
SelectWindow(MainPtr);        { and make window active            }
FrontWindow := MainPtr;       { remember that it's in front       }
DrawGrowIcon(MainPtr);        { draw the grow box in the corner   }
MainPeek := WindowPeek(MainPtr);{ get pointer to window record   }
MainPeek^.windowKind := UserKind; { window type = user kind(ID=8)}
ScreenArea := screenBits.Bounds; {get size of screen(don't assume)}
with ScreenArea do begin
  SetRect(DragArea,0,20,Right-5,Bottom-5);{ set drag region      }
  SetRect(GrowArea,90,50,Right,Bottom-40) { set grow region      }
end;
{ Now the program-specific initialization }
new(front);new(back);new(combo);
if ScreenArea.Right > 600 then
  begin
    MacII              := true;
    the_colour_choice := 2 ;{1or2..ColMacII}
    printEXoffset := 20;printWYoffset := 70;{suits US letter}
   end
else     { in page setup need to Expand 137% for MacII size:}
  begin
    MacII              := false;{ Expand 137% for MacII size.}
    the_colour_choice := 4 ;{ 4or5...the colour for monochrm MacSE}
    printEXoffset := 20; printWYoffset := 70;{suits US letter}
  end;
colourize(the_colour_choice) ; { set colour variables,line types  }
for i := 1 to 4 do begin front_EX_trans[i].ttype := tnone;
                         front_EX_trans[i].n     := 0.0 ;  end ;
front_WY_trans := front_EX_trans ;
back_EX_trans  := front_EX_trans ;
back_WY_trans  := front_EX_trans ;
save_EX_trans  := front_EX_trans ;
save_WY_trans  := front_EX_trans ;
combo_EX_trans := front_EX_trans ;
combo_WY_trans := front_EX_trans ;
back^.is_read     := false ;   { The back chart has not been read.}
back^.set_to_display := true ; { data array boolean for item check}
back^.showing     := false ;   { Used to make ticks go on and off.}
front^.is_read    := false ;   { The front chart has not been read}
front^.set_to_display := true ;{ data array boolean for item check}
front^.showing    := false ;   { Used to make ticks go on and off.}
combo^.is_read    := false ;   { When charts combined, this true  }
combo^.set_to_display := false;{ data array boolean for item check}
combo^.showing    := false ;   { Used to make ticks go on and off.}
this_front_saved := false ;    { boolean for set item state        }
this_combo_saved := false ;    { boolean for set item state        }
percent_change   := 10 ;       { Coarse movement at first          }
liftback         :=  0 ;       { Set to overlay mode for starters }
prt_x_mv_pixel   :=  0 ;       { only becomes non zero at do_print}
prt_y_mv_pixel   :=  0 ;       { only becomes non zero at do_print}
overlaid         := true ;     { Set to overlay mode for starters }
info_up          := false ;    { graph information not displayed  }
list_up          := false ;    { not using data editing function  }
grid_up          := true ;     { grid on and will be plotted       }
axis_up          := true ;     { axis matching back chart data     }
wiped            := false ;    { used during move or resize        }
comb_just_put_up := false ;    { used to control screen menu       }
normal_comb   := false ;       { front, back records are combined }
nlog_comb   := false ;         { front, back records nlogcombined }
```

```
  zero_point.h      := 0 ;        { used to set other points to zero }
  zero_point.v      := 0 ;        { used to set other points to zero }
  cross_point       := zero_point;{ cross_point - used for erasing.  }
  back_weight       := 1.0 ;      { back weighting ratio during comb.}
  front_weight      := 1.0 ;      {front weighting ratio during comb.}
  EXwindowdots := screenarea.right -1;{ max values same as given by }
  WYwindowdots := screenarea.bottom - 43;{SizeWindow in HandleGrow. }
  ClearWindow(MainPtr);           { Empty the window.                }
  SizeWindow(MainPtr,EXwindowdots,WYwindowdots,True);{resize window }
  DrawGrowIcon(MainPtr);          { draw the grow box in the corner  }
  centimetre_grid ;               { Prepare the screen - draw  grid  }
  back^.EXrange := useEXscreen * EXwindowdots ;
  back^.WYrange := useWYscreen * WYwindowdots ;
  window_EX_scale := useEXscreen * EXwindowdots / back^.EXrange ;
  window_WY_scale := useWYscreen * WYwindowdots / back^.WYrange ;
  prev_EX_scale := window_EX_scale ;{ compare scale change in grow. }
  prev_WY_scale := window_WY_scale ;{ compare scale change in grow. }
  VFlag := False ;
  FileSelected := False;             { set file opened false          }
  any_changes_to_file :=false;
  New(SPtr);
  UpdateMenu;                        { update menu as needed          }
  Finished := False  ;             { set program terminator to false }
end; { of proc Initialize }

procedure CleanUp;{to do whatever's needed before returning to Findr}
begin
  DisposeWindow(MainPtr);         { get rid of the main window       }
end; { of proc CleanUp }

procedure CursorAdjust;    { change cursors depending upon location }
var
  MousePt       : Point;
begin
  if MainPtr = FrontWindow then with MainPeek^ do begin
    GetMouse(MousePt);                       {  find where mouse is   }
    if PtInRect(MousePt,port.portRect) then {  if over window then   }
      if Button                              {   if button down...    }
        then SetCursor(CursList[plusCursor]^^) { then make a plus     }
        else  SetCursor(CursList[crossCursor]^^){ else make a cross   }
      else SetCursor(Arrow)                  {   else make an arrow   }
  end
end; { of proc CursorAdjust }

procedure HandleEvent(theEvent:EventRecord);{decode event,handle it.}
begin
  case theEvent.What of
    mouseDown      : DoMouseDown(theEvent);  { mouse button pushed   }
    mouseUp        : DoMouseUp(theEvent);    { mouse button released }
    keyDown        : DoKeyPress(theEvent);   { key pressed down      }
    autoKey        : DoKeyPress(theEvent);   { key held down         }
    updateEvt      : DoUpdate(theEvent);     { window need updating  }
    activateEvt    : DoActivate(theEvent)    { window made act/inact }
  end
end; { of proc HandleEvent }

begin { main body of program  }
   Initialize;                              { set everything up         }
   repeat                                   { keep doing the following }
     SystemTask;                            { update desk accessories  }
     CursorAdjust;                          { update which cursor       }
     if GetNextEvent(everyEvent,theEvent) { if there's an event...  }
       then HandleEvent(theEvent)          {     ...then handle it     }
   until Finished;                          {  until user is done      }
   Cleanup                                  {  clean everything up     }
end. { of program }
```

DISTRIBUTION

	Number of copies
DEPARTMENT OF DEFENCE	
Defence Science and Technology Organisation	
Chief Defence Scientist)	1
First Assistant Secretary Science Policy)	Shared Copy
Assistant Chief Defence Scientist (Operations))	for circulation
Counsellor Defence Science, London	Control Sheet Only
Counsellor Defence Science, Washington	Control Sheet Only
Electronics Research Laboratory	
Director, Electronic Research Laboratory	1
Chief, Electronic Warfare Division	1
Research Leader, Electronic Countermeasures	1
Head, Electronic Countermeasure Group	1
Dr Terry Moon, Electronic Countermeasure Group	1
Dr Peter Gerhardy, Electronic Countermeasure Group	1
Mr Peter Denton, Electronic Countermeasure Group	1
Mr Peter Bawden, Electronic Countermeasure Group	1
Surveillance Research Laboratory	
Director, Surveillance Research Laboratory	1
Chief, Microwave Radar Division	1
Dr Cheng Anderson (MRD)	1
Mr Robert Triggs (MRD)	1
Navy Office	
Navy Scientific Officer	Control Sheet Only
Army Office	
Scientific Adviser - Army	Control Sheet Only
Air Office	
Air Force Scientific Officer	Control Sheet Only
Joint Intelligence Organisation (DSTI)	1
Libraries and Information Services	
Librarian, Technical Reports Centre, Defence Central Library, Campbell Park	1
Document Exchange Centre Defence Information Services and Science Liaison Branch (for microfiche copying then destruction)	1
Main Library, Defence Science and Technology Organisation Salisbury	2
Library , Aeronautical Research Laboratories	1
Library, Materials Research Laboratories	1
Librarian, Defence Signals Directorate	1
Author	1

National Library of Australia	1
Australian Defence Force Academy Library	1
United States Defense Technical Information Centre	12
United Kingdom Defence Research Information Centre	2
Director Scientific Information Services, Canada	1
New Zealand Ministry of Defence	1
British Library, Document Supply Centre (UK)	1
Institution of Electrical Engineers (UK)	1
Engineering Societies Library (USA)	1
Spares	4
Total number of copies	47

DOCUMENT CONTROL DATA SHEET

Security classification of this page : UNCLASSIFIED

1 DOCUMENT NUMBERS	**2 SECURITY CLASSIFICATION**
AR Number : AR-005-990	a. Complete Document : Unclassified
Series Number : ERL-0505-GD	b. Title in Isolation : Unclassified
Other Numbers :	c. Summary in Isolation : Unclassified
	3 DOWNGRADING / DELIMITING INSTRUCTIONS

4 TITLE

GRAPH MATCH PROGRAM

5 PERSONAL AUTHOR (S)	**6 DOCUMENT DATE**
Mark T. Barrington	March 1990
	7 7.1 TOTAL NUMBER OF PAGES 50
	7.2 NUMBER OF REFERENCES 2
8 8.1 CORPORATE AUTHOR (S)	**9 REFERENCE NUMBERS**
Electronics Research Laboratory	a. Task : DST 4B/3LJ
	b. Sponsoring Agency :
8.2 DOCUMENT SERIES and NUMBER General Document 0505	**10 COST CODE**
11 IMPRINT (Publishing organisation)	**12 COMPUTER PROGRAM (S)** (Title (s) and language (s))
Defence Science and Technology Organisation Salisbury	

13 RELEASE LIMITATIONS (of the document)

Approved for Public Release.

Security classification of this page : UNCLASSIFIED

Security classification of this page : UNCLASSIFIED

14 ANNOUNCEMENT LIMITATIONS (of the information on these pages)

No limitation

15 DESCRIPTORS

16 COSATI CODES

a. EJC Thesaurus Terms	Graphs (Charts) Radar cross sections Comparison Computer programs	1709
b. Non - Thesaurus Terms		

17 SUMMARY OR ABSTRACT
(if this is security classified, the announcement of this report will be similarly classified)

Described herein is a program for manipulating and comparing graphs of radar cross-sections. The size and position of one graph relative to another can be changed and the two graphs combined to produce a new graph which represents an average of the two. The program can process data sets of varying numbers of data values. Graph Match was written using Apple Macintosh SE and II computers and Turbo Pascal as the programming language.

Security classification of this page : UNCLASSIFIED